

51单片机案例笔记

陈志旺 主 编



- 详尽剖析了22个典型案例
- 免费提供案例源程序 下载网址：www.cmpbook.com



机械工业出版社
CHINA MACHINE PRESS



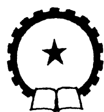
VISIT…

LANZAROTE Caliente.COM

51单片机案例笔记

陈志旺　主编

庞双杰　弓洪玮　马巨海　赵　鹏　等参编



机械工业出版社

本书以 51 系列单片机的原理及应用技术为内容，以自制的 51 仿真板为实践平台，精选了 22 个案例，介绍了 51 单片机的结构与原理、嵌入式系统开发流程、指令系统与程序设计、中断系统、定时/计数器、串行口、接口扩展、常用传感器等内容。本书结构安排简洁合理、主干清晰、层次分明、逻辑严谨、循序渐进，具有良好的可读性和操作性。

本书可作为单片机初学者的入门教程，也可作为高等院校机电工程、自动化、仪表测控等相关专业的单片机课程的课外读物，也可作为广大工程技术人员的参考用书。

图书在版编目（CIP）数据

51 单片机案例笔记 / 陈志旺主编．—北京：机械工业出版社，2015.4
ISBN 978-7-111-49736-3

Ⅰ. ① 5…　Ⅱ. ① 陈…　Ⅲ. ① 单片微型计算机—基本知识　Ⅳ. ① TP368.1

中国版本图书馆 CIP 数据核字（2015）第 057763 号

机械工业出版社（北京市百万庄大街 22 号　邮政编码 100037）
策划编辑：时　静　　责任校对：张艳霞
责任编辑：汤　枫
责任印制：李　洋

北京宝昌彩色印刷有限公司印刷

2015 年 4 月第 1 版 • 第 1 次印刷
184mm×260mm • 18.75 印张 • 459 千字
0001－3000 册
标准书号：ISBN 978-7-111-49736-3
定价：49.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务

服务咨询热线：（010）88379833

读者购书热线：（010）88379649

网络服务

机 工 官 网：www.cmpbook.com

机 工 官 博：weibo.com/cmp1952

教育服务网：www.cmpedu.com

金　书　网：www.golden-book.com

封面无防伪标均为盗版

前 言



在移动互联时代，嵌入式系统的作用日益凸显，而单片机被称为嵌入式系统的灵魂。虽然日新月异的科技发展使单片机的电子元器件密度越来越大、功能越来越强，但是初学者理想的入门单片机还是 51 系列。不仅因为 51 单片机曾经创造的辉煌，以至于现在许多电子产品还在使用 51 单片机，还因为当前主流的单片机常用的功能如串口、定时/计数器、中断等，8051 都具有。近年来，尽管 ARM 处理器凭借其低功耗等优势占领了移动处理器为主的嵌入式处理器大部分市场，但 51 单片机与之相比，开发环境、开发语言、硬件结构更简单，更适合初学者入门，而且在 51 单片机的基础上再学习 ARM 单片机会更有效率，因此本书仍以 51 系列单片机为例进行讲解。

《单片机原理及应用》在本质上是一门实践课，内容跨度大，知识点多，技能要求高，在有限的时间内难以充分掌握所有相关知识点，故在学习时，可选择少数应用实例，具体其需求，从上到下完成整个系统的分析设计，然后从下向上自行搭建整个系统，并在实践中领悟基本原理，准确掌握基本概念，培养和锻炼实际系统开发的技能，并最终设计和实现一个简单的小型嵌入式系统。本书在内容选材上精挑细选，尤其在案例枚举时，更是择优挑选、精益求精。所选的案例具有普适性和典型性。好的案例，可以以一当十，读者略加变通便能举一反三。另外，与其他案例书不同的是，不仅“精选案例”，而且还“深度解析”（即“笔记”部分的内容），这部分是与案例有关的精析内容，强调的是理解学习而不是机械记忆学习。

本书所应用的开发平台为编者开发的 51 单片机仿真板，经过燕山大学自动化系多年的生产实习应用，积累了许多经验，书中的许多内容就是来源于此。本书选用仿真板作为实践平台也体现了“实践是检验是否掌握单片机的唯一方法”这一理念。

另外针对初学者，本书强调“汇编语言”的基础作用，因为汇编语言直接与硬件相关，会对单片机的硬件学习有促进作用。因此本书的第 4 个案例，揭示了汇编语言指令与 51 单片机硬件的联系，这也是编者教学的独特心得。每个案例的软件代码还将汇编程序和 C 语言程序对比列出，便于读者学习。

本书结构安排简洁合理、主干清晰、层次分明、逻辑严谨、循序渐进，具有良好的可读性。编者始终贯彻启发性原则，用通俗易懂的语言叙述深奥的原理和复杂的术语，通过精选的案例说明概念，运用形象思维帮助学生建立感性认识，深入浅出地说明问题。

本书案例 4～18 和附录由燕山大学陈志旺编写，案例 19～22 由秦皇岛职业技术学院庞双杰编写，案例 2 由燕山大学弓洪纬编写，案例 3 由燕山大学马巨海编写，案例 1 由秦皇岛市第十五中学高级教师赵鹏编写，研究生陈林、张博强、陈晓、贺韶东、韩松、王泉策、李恒锐、王盼盼、高宁宁、王馨、徐少伟、常伯慧、张兴明、刘为宏、郝彬参与了本书部分程序的编写和书稿的校对工作，全书由陈志旺统稿。书中引用了一些网上文献，无法一一注明出处，在此向原作者表示感谢！

目　　录

前言
案例 1　解剖手机……1
1.1　案例任务……1
1.2　案例要点……1
1.3　案例设计……5
1.4　案例笔记……7
1.4.1　电子信息技术知识体系……7
1.4.2　嵌入式系统开发原则……8
案例 2　原理图和 PCB 图的绘制……11
2.1　案例任务……11
2.2　案例要点……11
2.3　案例设计……12
2.3.1　原理图绘制步骤……12
2.3.2　PCB 图绘制步骤……13
2.4　案例笔记……15
2.4.1　识读原理图的方法……15
2.4.2　识读 PCB 图的方法……16
2.4.3　框图、原理图、PCB 图、实物图的关系……17
2.4.4　51 单片机的芯片封装……18
2.4.5　如何保护电路设计的知识产权……21
案例 3　焊接与调试……22
3.1　案例任务……22
3.2　案例要点……22
3.3　案例设计……23
3.3.1　仿真板上的电气元件……23
3.3.2　仿真板焊接步骤……26
3.3.3　仿真板调试步骤……27
3.4　案例笔记……29
3.4.1　电源符号……29
3.4.2　接地的分类……29
3.4.3　干扰基础知识……30
3.4.4　如何判断集成电路的好坏……32
3.4.5　电气设备维修的十项原则……35
3.4.6　0Ω电阻……36
案例 4　指令系统的学习……37



4.1 案例任务 ······ 37
4.2 案例要点 ······ 37
4.3 案例设计 ······ 40
4.4 案例笔记 ······ 46
4.4.1 Keil C 的辅助调试功能 ······ 46
4.4.2 51 单片机中缩写的中英文速记 ······ 50
4.4.3 51 存储器总结 ······ 54
4.4.4 寻址方式总结 ······ 55
4.4.5 PSW 总结 ······ 55
4.4.6 有关 DPTR 的指令总结 ······ 56
4.4.7 控制转移指令总结 ······ 56
案例 5 程序的结构：分支和循环 ······ 58
5.1 案例任务 ······ 58
5.2 案例要点 ······ 58
5.3 案例设计 ······ 59
5.4 案例笔记 ······ 62
5.4.1 程序设计的方法 ······ 62
5.4.2 汇编语言常见错误总结 ······ 62
5.4.3 延时程序延时时间计算 ······ 63
案例 6 程序的效率 ······ 65
6.1 案例任务 ······ 65
6.2 案例要点 ······ 65
6.3 案例设计 ······ 66
6.4 案例笔记 ······ 69
6.4.1 汇编语言与 C 语言 ······ 69
6.4.2 C 语言常见错误总结 ······ 70
案例 7 流水灯 ······ 72
7.1 案例任务 ······ 72
7.2 案例要点 ······ 72
7.3 案例设计 ······ 73
7.3.1 硬件电路 ······ 73
7.3.2 软件代码 ······ 73
7.4 案例笔记 ······ 76
7.4.1 51 流水灯电路在 C51 学习中的应用 ······ 76
7.4.2 C51 双向口和准双向口 ······ 80
7.4.3 单片机端口驱动能力详解 ······ 80
7.4.4 LED 结构及发光原理 ······ 82
案例 8 输出模拟量的 I/O 端口 ······ 84
8.1 案例任务 ······ 84

8.2 案例要点 …… 84
8.3 案例设计 …… 84
8.3.1 硬件电路 …… 84
8.3.2 软件代码 …… 85
8.4 案例笔记：51PWM 与平均电压 …… 87
案例 9 蜂鸣器 …… 88
9.1 案例任务 …… 88
9.2 案例要点 …… 88
9.3 案例设计 …… 89
9.3.1 硬件电路 …… 89
9.3.2 软件代码 …… 89
9.4 案例笔记 …… 91
9.4.1 51 蜂鸣器结构及发声原理 …… 91
9.4.2 8051 晶体管驱动负载的技巧 …… 92
9.4.3 开关晶体管使用误区 …… 93
案例 10 并口的扩展 …… 95
10.1 案例任务 …… 95
10.2 案例要点 …… 95
10.3 案例设计 …… 97
10.3.1 硬件电路 …… 97
10.3.2 软件代码 …… 97
10.4 案例笔记 …… 98
10.4.1 8255 各工作方式总结 …… 98
10.4.2 微机扩展 I/O 接口的基础知识 …… 99
10.4.3 I/O 接口的数据传送方式 …… 100
10.4.4 可编程芯片总体要求 …… 101
案例 11 方式可控的流水灯 …… 102
11.1 案例任务 …… 102
11.2 案例要点 …… 102
11.3 案例设计 …… 102
11.3.1 硬件电路 …… 102
11.3.2 软件代码 …… 103
11.4 案例笔记：分支结构的常见错误 …… 104
案例 12 数字显示器 …… 107
12.1 案例任务 …… 107
12.2 案例要点 …… 107
12.3 案例设计 …… 107
12.3.1 硬件电路 …… 107
12.3.2 软件代码 …… 108

12.4　案例笔记：51 LED 显示码便于移植的解决方法……109
案例 13　行列键盘……112
13.1　案例任务……112
13.2　案例要点……112
13.3　案例设计……113
13.3.1　硬件电路……113
13.3.2　软件代码……114
13.4　案例笔记：C51 精炼的判键程序……121
案例 14　中断……124
14.1　案例任务……124
14.2　案例要点……124
14.3　案例设计……126
14.3.1　硬件电路……126
14.3.2　软件代码……127
14.4　案例笔记……130
14.4.1　利用 51 单片机的中断系统实现三级以上中断嵌套……130
14.4.2　中断与调用子程序的区别……131
案例 15　计数器……132
15.1　案例任务……132
15.2　案例要点……132
15.3　案例设计……134
15.3.1　硬件电路……134
15.3.2　软件代码……134
15.4　案例笔记：定时/计数器四种工作方式总结……137
案例 16　定时秒表……139
16.1　案例任务……139
16.2　案例要点……139
16.3　案例设计……139
16.3.1　硬件电路……139
16.3.2　软件代码……140
16.4　案例笔记……145
16.4.1　51 单片机定时计数器工作方式总结……145
16.4.2　C51 精确延时……147
16.4.3　中断与定时/计数器综合应用……149
16.4.4　实用技巧之扩展中断源……149
案例 17　频率显示器……150
17.1　案例任务……150
17.2　案例要点……150
17.3　案例设计……151

17.3.1 硬件电路 …… 151
17.3.2 软件代码 …… 151
17.4 案例笔记 …… 155
17.4.1 频率的测量方法 …… 155
17.4.2 利用门控制位 GATE 测量脉冲宽度 …… 156
案例 18 电压监控器 …… 157
18.1 案例任务 …… 157
18.2 案例要点 …… 157
18.3 案例设计 …… 158
18.3.1 硬件电路 …… 158
18.3.2 软件代码 …… 158
18.4 案例笔记 …… 168
18.4.1 逐次逼近式 A-D 转换 …… 168
18.4.2 A-D 转换器选择原则 …… 169
案例 19 温度监控器 …… 170
19.1 案例任务 …… 170
19.2 案例要点 …… 170
19.3 案例设计 …… 170
19.3.1 硬件电路 …… 170
19.3.2 软件代码 …… 171
19.4 案例笔记 …… 178
19.4.1 单总线 …… 178
19.4.2 DS18B20 总结 …… 181
案例 20 数字电子钟 …… 184
20.1 案例任务 …… 184
20.2 案例要点 …… 184
20.3 案例设计 …… 184
20.3.1 硬件电路 …… 184
20.3.2 软件代码 …… 184
20.4 案例笔记 …… 219
20.4.1 如何阅读英文的芯片数据手册 …… 219
20.4.2 DS1302 总结 …… 221
案例 21 串行通信 …… 224
21.1 案例任务 …… 224
21.2 案例要点 …… 224
21.3 案例设计 …… 226
21.3.1 硬件电路 …… 226
21.3.2 软件代码 …… 226
21.4 案例笔记 …… 233

21.4.1　SPI、I^2C、UART 三种串行总线协议的区别 …… 233
21.4.2　串口四种工作方式总结 …… 234
案例 22　数字计算器 …… 235
22.1　案例任务 …… 235
22.2　案例要点 …… 235
22.3　案例设计 …… 236
22.3.1　硬件电路 …… 236
22.3.2　软件代码 …… 237
22.4　案例笔记：电子产品设计步骤 …… 282
附录　51 单片机实验板原理图 …… 286
参考文献 …… 287



案例1



解 剖 手 机



1.1 案例任务

拆卸家里废弃的手机等嵌入式系统产品，找到其中的单片机，从互联网上了解一下此款单片机的功能及组成；在手机主板上找到相关外设芯片，识别其封装，找到其型号，并从互联网上了解其功能；找到主板上的电阻、电容、电感等电路元件，识别其封装，尽量通过其型号信息或万用表获知其电气参数；通过主板了解 PCB 布线的相关技巧。

1.2 案例要点

（1）计算机系统组成

计算机系统由计算机、外设、软件等组成。

（2）冯·诺依曼体系结构

计算机系统由硬件系统和软件系统两大部分组成。美籍匈牙利科学家冯·诺依曼（von Neumann）结构奠定了现代计算机的基本结构。

1）采用二进制形式表示数据和指令。

2）采用存储程序方式。这是冯・诺依曼思想的核心内容。它意味着事先编制程序，并将程序（包含指令和数据）存入主存储器中，计算机在运行程序时就能自动地、连续地从存储器中依次取出指令并执行。这是计算机能高速自动运行的基础。

3）由运算器、存储器、控制器、输入接口和输出接口五大部件组成计算机系统，并规定了这五部分的基本功能。

（3）电子产品拆卸、组装、维修相关知识

（4）元器件封装形式的识别

（5）电子产品常用元器件的识别

1）基本工具：电子产品制作基本工具见表 1-1。

表 1-1　电子产品制作基本工具

机械加工	基本：螺钉旋具、镊子、剪刀、裁纸刀、尖嘴钳、电钻 扩展：螺钉旋具套装、防静电镊子、刻刀、锉刀、台钳、手锯、斜口钳、剥线钳
材料粘贴	基本：电工胶布、透明胶、双面胶、502 快干胶 扩展：白乳胶、补鞋胶、万能胶
电路焊接	基本：电烙铁、烙铁架、焊锡丝、焊锡膏、吸锡器、空心针、高温海绵 扩展：助焊工具、焊接台、调温烙铁焊台
测量调试	基本：钢尺、卷尺、数字万用表 扩展：感应电笔、电子秤、舵机测试器

2）常用电路焊接工具：电工常用工具图示及功能见表 1-2。

表 1-2　电工常用工具图示及功能

实　物　图	简　　介
IC 拔取夹	用于将集成电路芯片从芯片座上取下而不弄弯引脚
镊子	用于贴片元件或螺钉的取用、夹持导线和元件，在焊接时夹持器件兼有散热作用。注意不可夹酸性药品；用完后必须使其保持清洁
剥线钳	用来剥掉细缆导线外部的绝缘层
尖嘴钳	用来取代手指折弯细金属丝，用来在狭窄的空间夹持螺母或其他小零件等，还可以用刀口剪断较硬的电线或细金属丝，但不能用于剪断较粗的金属丝，以防止将尖嘴钳损坏
偏口钳	用来剪断较粗的电线或细金属丝，修剪焊接后多余的线头，捋掉导线外层的绝缘皮等



（续）

实　物　图	简　　介
一字螺钉旋具	用来拧转螺钉以迫其就位的工具。选用一字螺钉旋具时，要注意螺钉旋具的刀口宽窄要与螺钉的一字槽相适应，既不能过长，也不能过厚，但也不能太薄。当刀口尺寸过长时，容易损坏安装件（对沉头螺钉）；当刀口的尺寸厚度超过螺钉的一字槽厚度时，可能会损坏螺钉槽
十字螺钉旋具	用螺钉旋具进行紧固和拆卸螺钉时，推压和旋转应同时进行，但在推压和旋转时不能用力过猛，以免损坏螺钉槽口
免焊万用电路板（面包板）	俗称面包板，内部是由一些长条形的磷青铜片组成，水平由 25 个插孔组成，而垂直线则是每 5 个插孔为一组，各插孔间可视需求，以 0.6mm 的单心线加以连接组合
万用电路板	和免焊万用电路板作用类似，但元件连接需焊接

（续）

实 物 图	简 介
手电钻	手电钻主要用来在金属板、电路板或机壳上打孔
钻头	和手电钻配合使用，根据钻孔大小可选不同规格
钢锯	用于精度不高、较硬材料的切割
锉刀	可用来锉平机壳开孔、金属板或绝缘板的毛边以及锉掉电烙铁头上的氧化物等
小型台虎钳	用来夹紧各种加工件，以便割锯、锉削和打孔等

（续）

实物图	简介
热熔胶枪	热熔胶枪是一种专门用来加热熔化热熔胶棒的专用工具。热熔胶枪内部采用居里点≥280℃的 PTC 陶瓷发热元件，并配设紧固导热结构，当热熔胶棒在加热腔中被迅速加热熔化为胶浆后，用手扣动扳机，即从喷嘴中挤出胶浆，供直接粘固用
绝缘套管	使导线焊接部绝缘
铜柱	电路板支撑用

▷▷ 1.3 案例设计

进行拆卸前首先要从互联网上搜集手机的相关资料，越全面越好，图 1-1 所示为 iPhone 5 的器件清单图。

图 1-2 所示为 iPhone 5 主板正视图，图中含有 iPhone 5 的“大脑”A6 处理器，其放大图如图 1-3 所示。A6 处理器的 CPU 性能和显卡性能均为 A5 的两倍，但是核心面积却缩小了 22%。配备 A6 处理器的 iPhone 5 应用加载速度非常快，例如，Page 加载速度达到之前的 2.1 倍，Keynote 加载速度也有之前的 1.7 倍。A6 处理器对摄像头拍照性能也有提升，A6 具有下一代 ISP、空间降噪、智能过滤、更好的低光照表现和更快的图像捕捉能力，拍照速度比 iPhone 4S 提升 40%。

iPhone 5 主板的背视图如图 1-4 所示，从中可以看到 Nand Flash 存储器，其型号和参数为海力士 H2JTDG2MBR 128 GB。

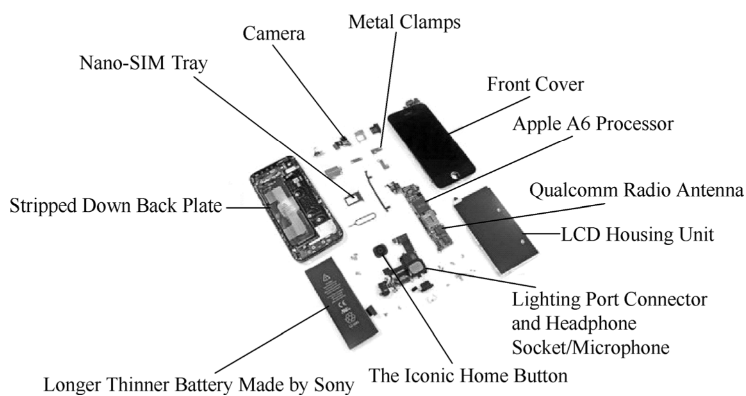


图 1-1　iPhone 5 器件清单

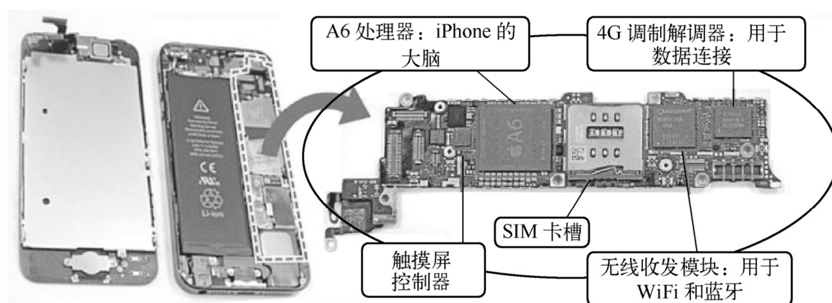


图 1-2　iPhone 5 主板正视图



图 1-3　A6 处理器

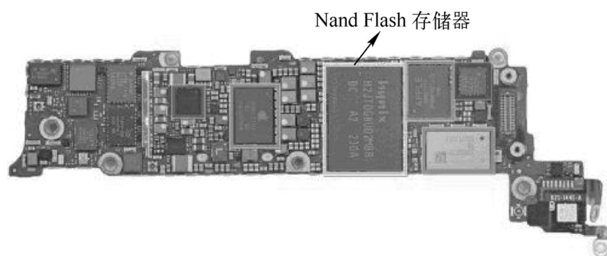


图 1-4　iPhone 5 主板背视图

拆机前要进行如下准备工作：

1）在手机内部的印制电路板上，由于镶嵌着不同生产厂商和不同型号的集成芯片，另外印制电路板上还有一些新型的元器件，所有这些芯片和元器件上都要传输一些弱电流信号，因此若手机还要使用，就不要在强磁场高电压下进行维修操作，以免遭大电流冲击，损坏元器件。



2）确认手机螺钉型号，准备专用的螺钉旋具；准备一个小的镊子和撬棒；准备一个取卡针；保持工作台面整洁，拆下的零部件要用一个专用的容器装好，避免散失。

3）拆机前建议佩戴好防静电手腕、接地线、防静电垫，以免因静电而造成手机内部电路的损坏，同时保证双手干燥。

4）拆机前确认手机已经完全关机。拆卸前应先取下电池、SIM 卡后再进行拆卸工作。

5）对手机中的重要信息进行备份。

拆机过程中，要注意如下事项：

1）在拆卸或者安装手机时，一定要按照一定的前后顺序进行操作，取放的芯片、元器件也要按一定的顺序排放，以免混淆或者丢失。

2）对于不易拆卸的手机，应先研究一下手机的外壳，看清手机壳的配合方式，然后再拆卸手机。

3）有些手机的固定螺钉十分隐蔽，如诺基亚 8810 手机推拉盖下有两颗螺钉，三星 N188 手机后壳防尘罩下的螺钉，拆卸前应仔细查找，在没有全部拆卸螺钉的情况下不要强行打开机壳，以免对机壳造成不可修复的损伤。将手机中的相关固定螺钉全部取出（部分螺钉较隐蔽，要细心查找），某些机器还要把天线或其他配件取下。

4）取下主板的过程中，要留意是否有主板与外壳相连接的排线，如果有，须先撬开排线头，然后再取外壳，以免损坏排线。

5）显示屏为易损器件，要轻取轻放，不能用力过大。

6）准备一部数码相机或者拍照清晰度较高的手机，用来记录拆机细节，方便重新安装手机时参考。

在拆卸中若出现螺钉滑丝，可在螺钉旋具上垫一些东西（如双面胶、布料等），可以增加螺钉旋具与螺母间的摩擦力；若螺母是露出外面的，可以用尖嘴钳夹紧螺钉后，旋出螺钉。

▷▷ 1.4 案例笔记

▷▷▷ 1.4.1 电子信息技术知识体系

可以把电子信息科学技术的知识架构类比于生物系统，如图 1-5 所示。生物系统从基本的原子、分子形成蛋白质、细胞器、细胞、组织、器官、系统到个体的人，在不同的层次上形成不同的生物，有低等生物和高等生物、植物和动物，每一层次各自独立又相互联系，有着从低到高的递进关系，低层次生物的进化决定了高层次生物的特性和发展。电子信息科学技术的知识体系与此类似。电子信息学科从基础物理发展而来，从电磁场和电荷载体的相互作用开始，到电势（电流、电压）和电路之间的相互关系；从电路里分化出逻辑电路，从电

势与电路的关系，到比特和逻辑之间的相互关系；在逻辑电路的基础上研制出中央处理器，从比特与逻辑的关系，发展到指令集和处理器的相互关系；给中央处理器包装上操作系统以后，有了计算机，在计算机这个层面上不再讨论具体的处理器，而是讲数据和算法；计算机互联形成网络，数据包和网络的相互关系又有质的飞跃；这样一直到人的大脑处理的各种媒体，形成认知和媒体的关系。这几个层次实际上构成了电子信息科学技术整个知识的脉络，而且每一个层次对问题的描述都有质的、革命性的变化。整个体系从基本系统到多功能复杂系统，与生物系统的变革非常类似。

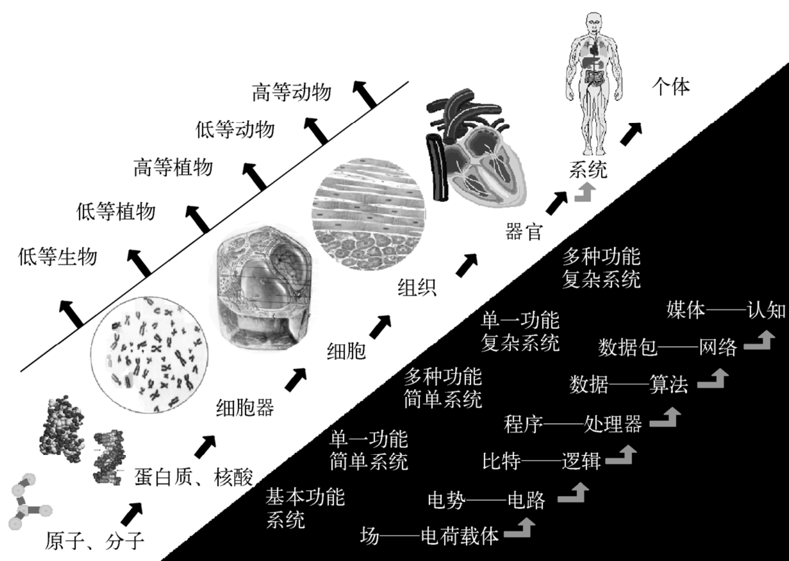


图 1-5　电子信息科学技术的知识架构和生物系统的类比

电子信息科学技术的知识架构如图 1-6 所示，知识的核心概念可以归纳在“信息载体与系统的相互作用”这一整体脉络下。信息载体是携带信息的，信息不能离开载体而存在。电子信息科学技术涉及的载体有电磁场、电流/电压、比特、CPU 指令集、计算机的数据、网络上的数据包、包括人看到的媒体等形式。不同的信息载体有不同的系统与之相互作用，相应的有物质、电路、处理器、算法、网络、人的大脑等。如图 1-6 所示，从基础的数学物理开始，围绕几个层次的信息载体与系统相互作用，具体来说是场和电子、电势与电路、比特与逻辑、程序与处理器、数据与算法、媒体和认知，把整个学科的核心概念整合到了一起。每一个层次上都是不同的信息载体与系统的相互作用，各层次之间相互关联，逐次递进。

计算机技术是电子信息科学技术的核心，嵌入式系统产品是电子信息科学技术的典型应用。

▷▷▷ 1.4.2　嵌入式系统开发原则

（1）硬件设计

一个嵌入式系统的硬件电路设计包括三部分内容：单片机芯片的选择、单片机系统扩展、系统配置。

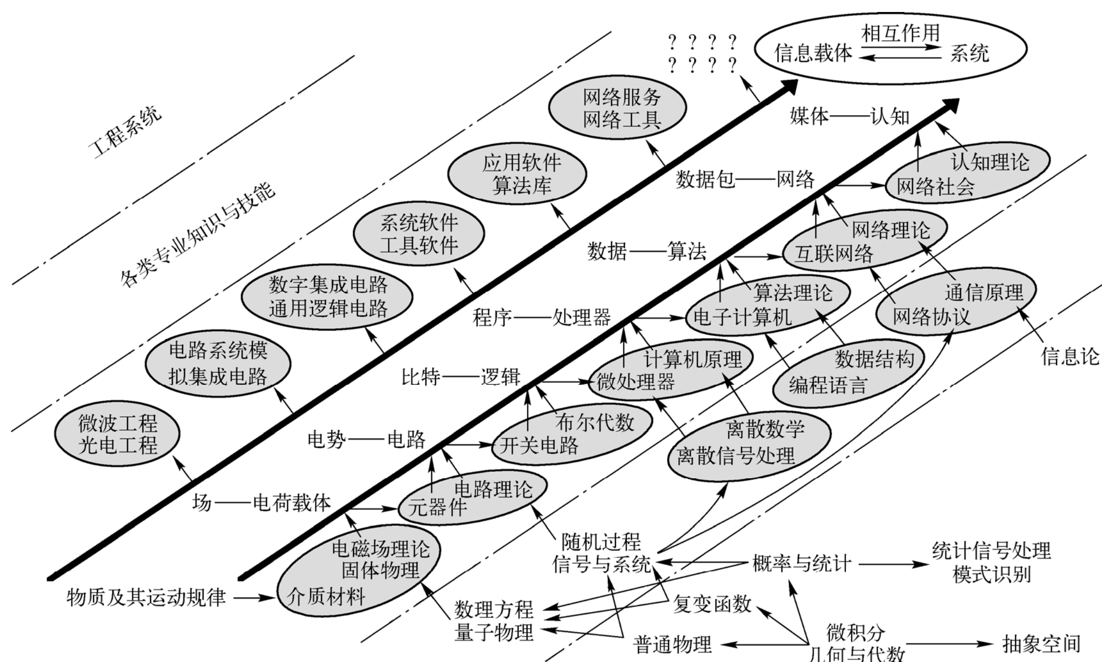


图 1-6 电子信息科学技术的知识架构

单片机系统扩展是指单片机内部的功能单元（如程序存储器、数据存储器、I/O 口、定时器/计数器、中断系统等）的容量不能满足应用系统的要求时，必须在片外进行扩展，这时应选择适当的芯片，设计相应的扩展连接电路。

系统配置是按照系统功能要求配置外设，如键盘、显示器、打印机、A-D 转换器、D-A 转换器等，设计相应的接口电路。 系统扩展和配置设计遵循的原则为：

1）尽可能选择典型通用的电路，并符合单片机的常规用法。

2）系统的扩展与外设配置的水平应充分满足应用系统当前的功能要求，并留有适当余地，便于以后进行功能的扩充。

3）硬件结构应结合应用软件方案一并考虑。

4）整个系统中相关的元器件要尽可能做到性能匹配。

5）可靠性及抗干扰设计是硬件设计中不可忽视的一部分。

6）单片机外接电路较多时，必须考虑其驱动能力。

（2）软件设计

一个应用系统中的软件一般是由系统监控程序和应用程序两部分构成的。其中，应用程序是用来完成诸如测量、计算、显示、打印、输出控制等各种实质性功能的软件；系统监控程序是控制单片机系统按预定操作方式运行的程序，它负责组织调度各应用程序模块，完成系统自检、初始化、处理键盘命令、处理接口命令、处理条件触发和显示等功能。

软件设计时，应根据系统软件功能要求，将软件分成若干个相对独立的部分，并根据它们之间的联系和时间上的关系，设计出软件的总体结构，画出程序流程框图。画流程框图时还要对系统资源作具体的分配和说明。根据系统特点和用户的了解情况选择编程语言，现在一般用汇编语言和 C 语言。汇编语言编写程序对硬件操作很方便，早期的单片机应用系统软

件主要用汇编语言编写；C 语言功能丰富，表达能力强，使用灵活方便，应用面广，目标程序效率高，可移植性好，现在单片机应用系统开发很多采用 C 语言来进行开发和设计。

一个优秀的应用系统的软件应具有以下特点：

1）软件结构清晰、简捷、流程合理。

2）各功能程序实现模块化、系统化。这样，既便于调试、连接，又便于移植、修改和维护。

3）程序存储区、数据存储区规划合理，既能节约存储容量，又能给程序设计与操作带来方便。

4）运行状态实现标志化管理。各个功能程序运行状态、运行结果以及运行需求都设置状态标志以便查询，程序的转移、运行、控制都可通过状态标志来控制。

5）经过调试修改后的程序应进行规范化，除去修改“痕迹”。规范化的程序便于交流、借鉴，也为今后的软件模块化、标准化打下基础。

6）实现全面软件抗干扰设计。软件抗干扰是计算机应用系统提高可靠性的有力措施。

7）为了提高运行的可靠性，在应用软件中设置自诊断程序，在系统运行前先运行自诊断程序，用以检查系统各特征参数是否正常。

案例 2



原理图和 PCB 图的绘制



2.1 案例任务

1）利用 Altium Designer 软件完成 51 单片机开发板原理图的绘制。
2）利用 Altium Designer 软件完成 51 单片机开发板 PCB 图的绘制。
3）拆卸家里废弃的电器产品，观察 PCB 上有哪些地方符合 PCB 图的绘制规律。

2.2 案例要点

1）Altium Designer（缩写为 AD）的安装与使用。
2）原理图和 PCB 图的重要概念。
原理图和 PCB 图的关系如图 2-1 所示。

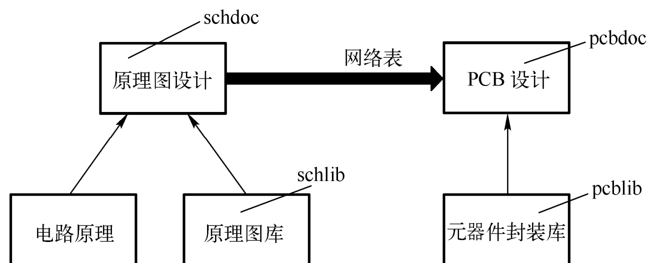


图 2-1　原理图和 PCB 图的关系

图 2-1 中原理图库和元器件封装库提供各种元器件在电路图上的形状。

实际元器件在原理图和 PCB 图中的符号示例见表 2-1。同一个元器件，可以有不同的原理图符号对应同一种元器件，可以有很多种各异的封装形式。

表 2-1　实际元器件在原理图和 PCB 图中的符号

实际元器件	原理图符号	元器件封装
	Q? NPN	Q? NPN

3）掌握 AD 中原理图绘制的步骤，即新建原理图、图纸设置、装载原理图元器件库、放置元器件、连线、注解、检查修改和打印输出。

4）能阅读简单的电气设备原理电路图。

5）掌握利用元器件资料绘制一个元器件封装。

6）掌握 AD 中 PCB 图绘制的步骤，即新建 PCB 图、图纸设置、规则设置、装载网络表和元器件封装、布局、布线、敷铜和设计规则检查（DRC）。

7）掌握元器件布局的方法。

8）理解主要布线规则的含义。

9）掌握主要布线规则的设置方法、自动布线的操作方法。

2.3 案例设计

2.3.1 原理图绘制步骤

电路原理图是详细说明电子元器件相互之间、电子元器件与单元电路之间、产品组件之间的连接关系，以及电路各部分电气工作原理的图形。它是产品设计和性能分析的原始资料，也是编制印制电路板、装配图和接线图的依据。设计原理图之前需要理解电路的工作原理，正确利用计算公式，满足设计要求：

1）元器件的工作电流、电压、频率和功耗等参数应能满足电路指标的要求。

2）元器件的极限参数必须留有足够裕量，一般应大于额定值的 1.5 倍。

3）电阻器和电容器的参数应选计算值附近的标称值。

设计时要根据电路的要求选择性能和参数合适的阻容元件，并注意功耗、容量、频率和耐压范围是否满足要求。

原理图编制步骤如图 2-2 所示，可按下面过程来完成：

1）设置工作环境。设置 Schematic 设计环境，包括设置格点大小和类型、光标类型等，大多数参数也可以使用系统默认值。设计好图纸大小，图纸大小是根据电路图的规模和复杂程度而定的，设置合适的图纸大小是设计好原理图的第一步。

2）放置元器件。用户根据电路图的需要，将元器件从元器件库里取出放置到图纸上，并对放置元器件的序号、元器件封装进行定义和设定等工作。

3）原理图布线。利用 Schematic 提供的各种工具，将图纸上的元器件用具有电气意义的导线、符号连接起来，构成一个完整的原理图。将初步绘制好的电路图作进一步的调整和修改，使得原理图更加美观。

4）建立网络表。通过 Schematic 提供的各种报表工具生成报表，其中最重要的报表是网络表，通过网络表为后续

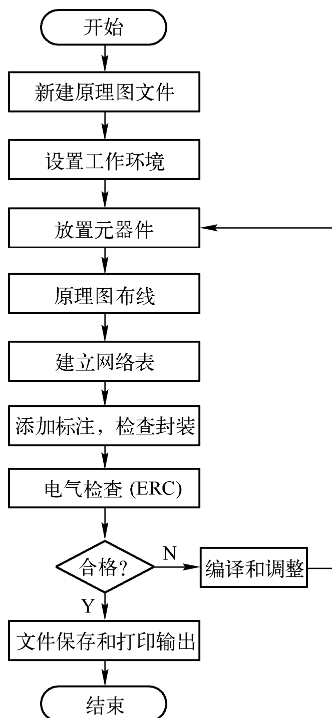


图 2-2　原理图绘制步骤

的电路板设计做准备。

5）添加标注，检查封装。

6）电气检查。

7）文件保存及打印输出。

51 单片机开发板的原理图详见本书附录。

一张好的原理图，不仅要求没有错误，还应该美观、信号流向清楚、标注清晰和可读性强。这就首先要求在绘制原理图时，应该遵循以下原则：

1）文字符号。原则上，图中所有元器件应以国家标准规定的图形符号和文字代号表示，文字代号一般标注在图形符号的右方或上方。

2）分图。有时一个总电路图由几部分组成，绘制时应尽量把总电路图画在一张纸上。如果电路比较复杂，需绘制几张图，则应把主电路图画在一张图纸上，而把一些比较独立或次要的单元电路画在其他图纸上，并在图的端口两端做上标记，标出信号从一张图到另一张图的引出点和引入点，以此说明各图纸在电路连线之间的关系。

3）元器件排列。注意信号的流向。顺着信号的流向摆放元器件；一般从输入端或信号源画起，从左到右或从上到下按信号的流向依次画出各单元电路，而反馈通路的信号流向则与此相反。同一个模块中的元器件靠近放置，不同模块的元器件稍远一些放置；串联的元器件最好画在一条直线上；并联时，各元器件符号的中心对齐。

4）连线。通常连接线可以水平布置或垂直布置，一般不画斜线，并且交叉和折弯应最少。互相连通的交叉线，应在交叉处用圆点表示。根据需要，可以在连接线上加注信号名或其他标记，表示其功能或其去向。一般不要从一点上引出多于三根的连线。有的连线可用符号表示，如元器件的电源一般标出电源电压的数值。

5）标注。在电路原理图中，各元器件文字符号的右下方都标有脚注序号，该脚注序号是按同类元件的多少来编制的，或者按照各元器件在图中的位置自左向右，或自上而下进行顺序编号的，一般情况下是用阿拉伯数字进行标注，如 R_1、R_2、C_4、C_5、VD_8、VD_9、V_8、IC_1、IC_2 等。

6）如果电子产品由几个单元电路组成，此时便可在各单元电路的元器件文字符号前面加一该单元电路的顺序号，如 $2R_1$、$3R_2$、$2C_4$、$3C_5$ 等。

7）在电路原理图中标出各元器件的具体型号和参数，为日后的检测与更换提供了依据。另外，在有的电路原理图中还标出了关键点直流工作电压值的大小、某些元件的额定功率、电压、电流等参数，以及部分电路的调试或安装条件，也为检测与维修提供了方便。

▷▷▷ 2.3.2 PCB 图绘制步骤

印制电路板图是用来表示各种元器件在实际电路板上的具体方位、大小以及各元件与印制电路板的连接关系的图样。印制电路板图是在电子产品组装、调试、检测时使用的。一般情况下，要与电路原理图配合使用，在电路原理图中的每个元器件、每条连线以及连接器等在印制电路板图中都有相应的位置，因此两者对照使用，便可较快地确定所要找的元器件的具体位置，从而给组装和维修带来极大的方便。

PCB 图的主要任务是根据电路的原理和所需元器件的封装形式进行物理结构的布局和布线。PCB 图绘制步骤如图 2-3 所示。

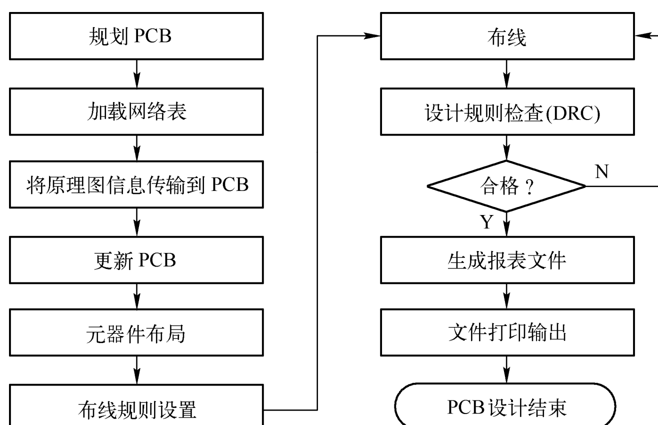


图 2-3　PCB 图绘制步骤

51 单片机开发板的 PCB 图如图 2-4 所示。

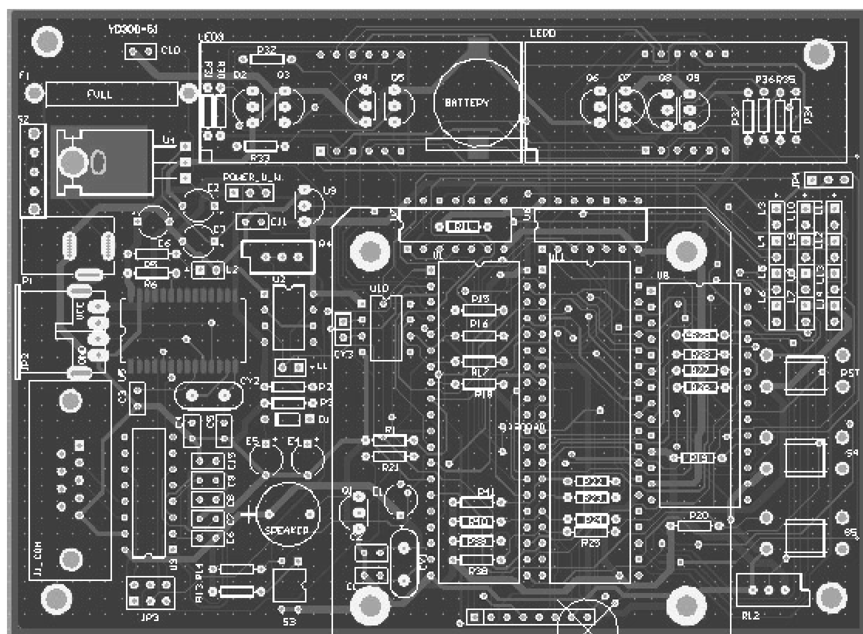


图 2-4　51 单片机开发板的 PCB 图

印制电路板一般在专业工厂进行制作。主要涉及的操作如下：

1）加工敷铜板。

2）根据电子工程师提供的印制电路板布线图（如 AD 软件系列中的*.PCB 文件）制作感光胶片，然后再腐蚀铜皮，只保留代表导线的部分及作为屏蔽体的部分。

3）压合。

4）镀通孔，上铜。对于金属化孔还要通过电镀方法沉积金属，形成金属化通孔。

5）外层线路上铜。

6）在表面给焊接孔涂敷助焊剂，其他部分镀上有阻焊和绝缘作用的覆盖层。

7）在元器件面印制标识元器件的字符。

8）成型切割。

9）终检包装。

2.4 案例笔记

2.4.1 识读原理图的方法

一张电路图就好像是一篇文章，各种单元电路就好比是句子，而各种元器件就是组成句子的单词。所以要想看懂电路图，还得从认识单词——元器件图形符号及文字符号开始。

元器件图形符号是用来表示实物的符号。在画电子产品电路原理图时，不能将电子元器件的实物图都画出来，如果画出实物图，那将使电路图变得很大且复杂，为此国家规定了统一的电路图形符号标准。我国曾先后几次颁发过电路图形符号标准，其图形符号不尽一致，为使科技工作者都能贯彻执行新标准，不致在使用中造成混乱，原国家标准局在《全国电气领域全面推行电气制图和图形符号国家标准的通知》中明确规定：自 1990 年 1 月 1 日起，所有电气技术文件、图样和书刊一律使用新的国家标准。熟悉电路图形符号是识读电路图的最基本要求，只有熟悉和认识电路图形符号才能读懂电路图，才能对电路图作进一步分析。

每一种元器件都有许许多多的应用，典型应用电路是最为常见的应用电路。为方便、快捷地看懂、读通电路图，还要掌握一些由常用元器件组成的单元电子电路知识，如整流电路、滤波电路、放大电路、振荡电路、电源电路等。因为这些单元电路是电子产品电路图中常见的功能块，掌握这些单元电路的知识，不仅可以深化对电子元器件的认识，而且通过这样的初级练习，也是对看懂、读通电路图的锻炼。有了这些知识，就为进一步看懂、读通较复杂的电路图奠定了良好的基础。

电子电路的主要任务是对信号进行处理，只是处理的方式（如放大、滤波、变换等）及效果不同而已。因此分析电路图时，应以所处理的信号流向为主线，沿信号的主要通路，以基本单元电路为依据，将整个电路分成若干具有独立功能的部分，并进行分析。具体步骤可归纳如下：

1）了解用途。指了解所读的电子电路原理图用于何处，起什么作用。这对于弄清工作原理、各部分的功能及性能指标都有指导意义。

2）找出通路。指找出信号流向的通路。一般的规律是输入在左方，输出在右方，电源在下方（也有不画出电源的）。信号传输的枢纽是有源器件，可以按它们的连接关系来找。通路找出后，电路的主要组成部分就显示出来了。这就是分析电路图的重点。

3）化整为零。沿信号的主要通路，将原理图分成若干具有单一功能的部分。划分的精细程度与读图者掌握电路类型的多少及经验有关。

4）分析功能。划分成单元电路后，根据已有的知识，定性分析每个单元电路的工作原理和功能。

5）统观整体。先将各部分的功能用相应的框图表示出来，可用文字、表达式、传输特性、信号波形等方式在框图中注出，然后根据它们之间的关系画成一个整体的框图。有了框图就可以看出各单元电路之间是如何互相配合来实现电路所具有的功能的。

6）性能估算。这是指对各部分电路的性能进行定量估算并进一步得出整个电路的性能指标，从而可以了解各部分对性能的影响并大致找到影响指标的主要环节，为调整、维修电路打下基础。

至此，电路的基本情况就大致清楚了，需要指出的是，对于不同水平的读图者或不同的电路，所采取的具体步骤可能是不一样的，上述方法仅供参考。至于电路中的次要部分和调整哪些元件的参数能改善哪些技术指标，以及对各部分电路的性能进行定量估算以进一步得出整个电路的性能指标等，则完全根据读图者的能力自行分析。

最后给出读图的口诀：弄清用途，化繁为简，抓住两头，找出电源。以管为主，从左到右，分析电位，揪住地线。抓住两头，是指抓住输入、输出两头，分析信号的输入回路和最后输出的控制对象；找出电源，是指搞清楚各部分所用电源电压的极性和大小以及它们的来源分析电位；揪住地线，是指分析元器件和某一节点的电位变化时，一定要以“共地线”为基准，否则就搞不清电位变化的趋向，这在分析模拟放大器负反馈作用中是非常重要的。

▷▷▷ 2.4.2　识读 PCB 图的方法

印制电路板对于维修工作有很重要的作用，因为通过印制电路板能比较快地找到需要检测的元件，加快维修速度，所以学会识读印制电路板非常重要。由于印制电路板上面的元器件排列没有什么规律可循，故不像识读电路原理图那样方便。下面介绍识读印制电路板的方法。

1）先找到醒目的元器件。因为印制电路板的走线（印制导线）有的地方粗，有的地方细，而且无规律，焊盘的形状有大有小，这样就给寻找某一个元器件的具体位置带来不便，为此比较醒目的元器件就成为寻找其他元器件的参考点。

由于晶体管、集成电路、可调电阻等的数量较电阻、电容少很多，而且电路符号及实物也比较容易识别，故以此类元器件的位置为醒目的参考点。因此识读印制电路板时，首先找到醒目的元器件，然后再去找其他元器件。

2）电路原理图与印制电路板相互对照。因为电路原理图与印制电路板元件编号是一致的，为能很快在印制电路板中找到所要测试的元器件，可以先在电路原理图中找到所需测试的元器件编号，然后再找此编号的元器件周围有何醒目的元器件。这样在印制电路板中就可比较容易地找到所需测量的元器件。

3）弄清印制电路板中单元电路的划分。有些印制电路板图是把单元电路中的元器件相对集中地放在一个小范围内，这样的印制电路板图对于查找元器件的具体位置相对容易，只要找到单元电路中的一个，其他的元器件就在其附近，查找起来很方便。

4）正确区分印制电路板的地线、电源线和信号线。以电源电路为例，电源变压器次级所接整流管的负端为电源正极，与地线之间一般均接有大容量滤波电容，该电容外壳有极性标志。也可从三端稳压器引脚找出电源线和地线。工厂在印制电路板布线时，为防止自激、抗干扰，一般把地线铜箔设置得最宽（高频电路则常有大面积接地铜箔），电源线铜箔次之，信号线铜箔最窄。此外，在既有模拟电路又有数字电路的电子产品中，印制电路板上往往将各自的地线分开，形成独立的接地网，这也可作为识别判断的依据。因偏置电阻、限流电阻、电源滤波电容等元件均与电源有直接的连接，所以沿着通有直流电流的印制导线便可较容易地找到所需检测的元器件。

▷▷▷ 2.4.3 框图、原理图、PCB 图、实物图的关系

描述电路的图示除了本案例所述的原理图和 PCB 图之外，还包括框图和实物图。下面以一个声音系统为例来论述它们之间的关系。声音系统实物图如图 2-5 所示，其 PCB 图如图 2-6 所示。

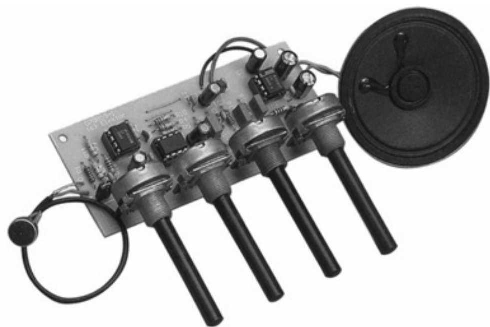


图 2-5 声音系统实物图

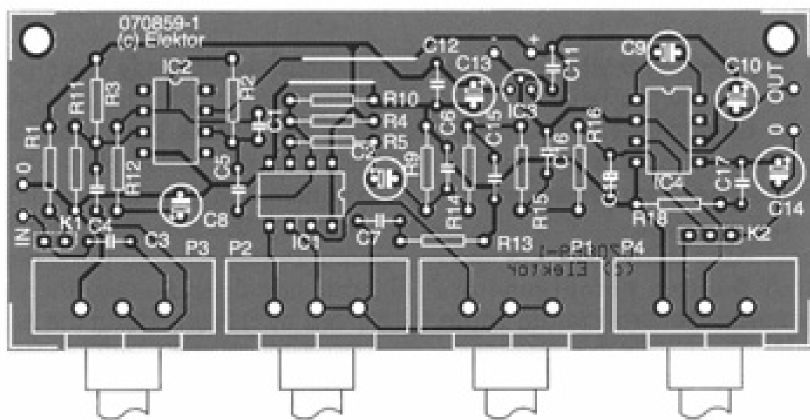


图 2-6 声音系统 PCB 图

声音系统框图如图 2-7 所示。框图主要是用一些方框和少量图形符号来表示的一种图样，它主要是体现电子产品各个组成部分以及它们在电性能方面所起作用的原理和信号的流程顺序。框图的特点如下：

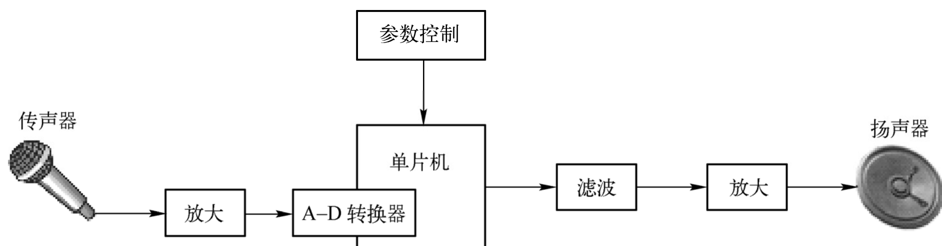


图 2-7 声音系统框图

1）各个组成部分自左向右或自上而下排成一列或数列。在矩形、正方形内或图形符号上按其作用标出它们的名称或代号。

2）各组成部分间的连接用实线表示，机械连接以虚线表示，并在连接线上用箭头表示其作用过程和方向。必要时可在连接线上方标注该处的特征参数，如信号电平、波形、频率和阻抗等。

绘制框图时，要在方框内使用文字或图形注明该方框所代表电路的内容或功能，方框之间一般用带有箭头的连线表示信号的流向。在框图中，也可以用一些符号代表某些元件，如天线、电容器、扬声器等。

对于复杂电路，框图可以扩展为流程图。在流程图里，“方框”成为广义的概念，代表某种功能而不管具体电路如何，“方框”的形式也有所改变。流程图实际是信息处理的“顺序结构”、“选择结构”和“循环结构”以及这几种结构的组合。

▷▷▷ 2.4.4 51 单片机的芯片封装

AT89S51 是单片机的一种型号，这种型号下有 DIP、PLCC、TQFP 等封装。

（1）DIP 封装

DIP（Dual In-line Package）是指采用双列直插形式封装的集成电路芯片，绝大多数中小规模集成电路（IC）均采用这种封装形式，其引脚数一般不超过 100 个。采用 DIP 封装的 CPU 芯片有两排引脚，需要插入到具有 DIP 结构的芯片插座上。当然，也可以直接插在有相同焊孔数和几何排列的电路板上进行焊接。DIP 封装的芯片从芯片插座上插拔时应特别小心，以免损坏引脚。

DIP 封装具有以下特点：

1）适合在 PCB 上穿孔焊接，操作方便。

2）芯片面积与封装面积之间的比值较大，故体积也较大。

51 单片机的 DIP 封装如图 2-8 所示。

（2）PLCC 封装

PLCC（Plastic Leaded Chip Carrier，带引线的塑料芯片载体）封装是表面贴装型封装之一，外形呈正方形，32 脚封装，引脚从封装的四个侧面引出，呈丁字形，是塑料制品，外形尺寸比 DIP 封装小得多。PLCC 封装适合用 SMT 表面安装技术在 PCB 上安装布线，具有外形尺寸小、可靠性高的优点。

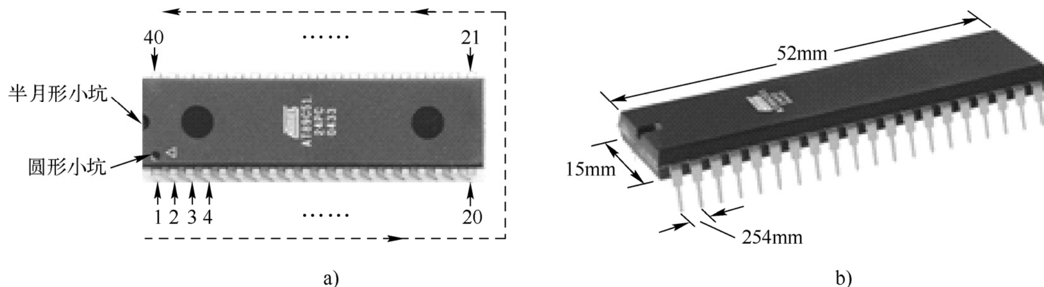
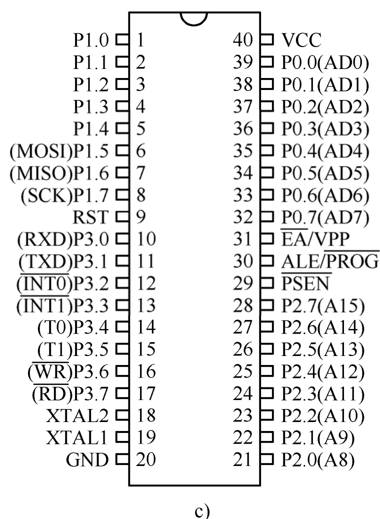
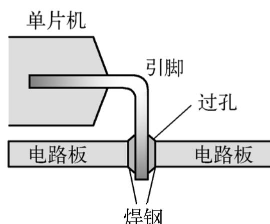


图 2-8 51 单片机的 DIP 封装

a) 器件外观 b) 器件外观及尺寸



c)



d)

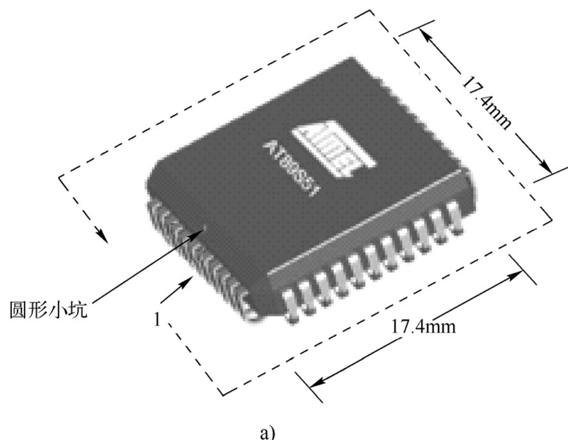
图 2-8　51 单片机的 DIP 封装（续）

c) 电路符号　d) 焊接剖面

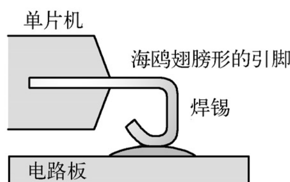
51 单片机的 PLCC 封装如图 2-9 所示。

（3）TQFP 封装

TQFP（Thin Quad Flat Package，薄塑封四角扁平封装）封装的芯片引脚之间距离很小，引脚很细，一般大规模或超大型集成电路都采用这种封装形式，其引脚数一般在 100 个以上。用这种形式封装的芯片必须采用 SMD（表面安装设备技术）将芯片与主板焊接起来。采用 SMD 安装的芯片不必在主板上打孔，一般在主板表面上有设计好的相应引脚的焊点。将芯片各引脚对准相应的焊点，即可实现与主板的焊接。用这种方法焊上去的芯片，如果不用专用工具是很难拆卸下来的。



a)



b)

图 2-9　51 单片机的 PLCC 封装

a) 器件外观　b) 焊接剖面

c)



图 2-9　51 单片机的 PLCC 封装（续）

c) 电路符号

TQFP 封装具有以下特点：

1）适用于 SMD 安装技术在 PCB 上安装布线。

2）适合高频使用。

3）操作方便，可靠性高。

4）芯片面积与封装面积之间的比值较小。

51 单片机的 TQFP 封装如图 2-10 所示。

a)　　b)

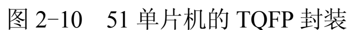
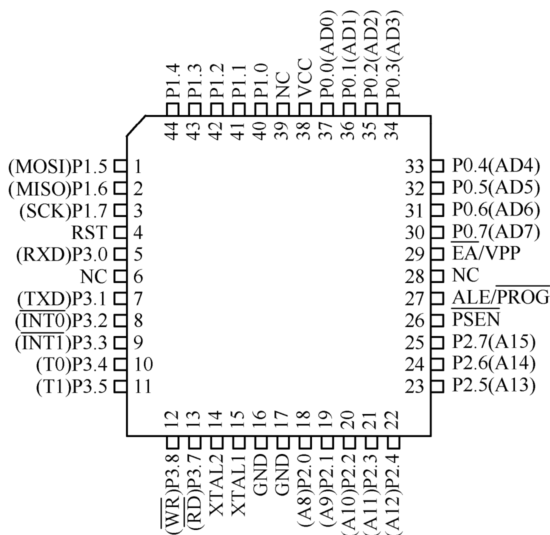


图 2-10　51 单片机的 TQFP 封装

a) 器件外观　b) 焊接剖面



c)

图 2-10　51 单片机的 TQFP 封装（续）

c) 电路符号

▷▷▷ 2.4.5　如何保护电路设计的知识产权

大多数工程师都习惯于将 PCB 文件设计好后直接送 PCB 工厂加工，而国际上比较流行的做法是将 PCB 文件转换为 GERBER 文件和钻孔数据后交 PCB 工厂，为何要“多此一举”呢？

因为电子工程师和 PCB 工程师对 PCB 的理解不一样，由 PCB 工厂转换出来的 GERBER 文件可能不是您所要的，如果在设计时将元器件的参数都定义在 PCB 文件中，而又不让这些参数显示在 PCB 成品上，未作说明时，PCB 工厂依葫芦画瓢将这些参数都留在了 PCB 成品上。这只是一个例子。若将 PCB 文件转换成 GERBER 文件就可避免此类事件发生。还有就是为了保护自己的劳动成果不被窃取，公司的机密不被盗窃。这才是 GERBER 文件的作用。

GERBER 文件是一种国际标准的光绘格式文件，包含 RS-274-D 和 RS-274-X 两种格式。常用的 CAD 软件都能生成这两种格式文件。

案例3



焊接与调试

▷▷3.1　案例任务

1）完成 51 单片机开发板的焊接任务。
2）完成 51 单片机开发板的调试任务。

▷▷3.2　案例要点

（1）熟悉常用的电阻、电容、电感、二极管、晶体管、芯片等元器件功能

（2）了解焊接 IPC 标准

国际电子工业联接协会（Association Connecting Electronics Industries）是一家全球性非盈利电子行业协会。其前身为印制电路协会（Institute of Printed Circuits，IPC），现在的协会仍然保持 IPC 标识。公司的业务之一是组织开发公司、客户及供应商的标准，确保产品能够按照此标准进行生产、检验和测量，以保证产品达到既定的可靠性和使用性能。IPC-A-600 既是对 PCB 制定的验收条件，也可对 PCB 产品生产过程的作业以及产品质量保证提供指导。

（3）熟练掌握焊接技能

能应用图 3-1 所示的设备焊接合格电子产品。



图 3-1　常用焊接设备

（4）了解调试的定义及作用

1）调试是用测量仪表和一定的操作方法按照调试工艺规定对单元电路板和整机的各个可调元器件或零部件进行调整与测试，使产品达到技术文件所规定的性能指标。

2）调试的作用包括：实现产品功能、保证质量的重要工序；发现产品设计、工艺缺陷和不足的重要环节；为不断提高产品的性能和品质积累可靠的技术性能参数。

（5）能正确使用常用的电工仪表、电子仪器及常用的电气设备

▷▷ 3.3 案例设计

▷▷▷ 3.3.1 仿真板上的电气元件

单片机开发板原理图见本书附录。

单片机开发板组成如图 3-2 和图 3-3 所示，各主要元器件名称见表 3-1。该实验平台基本上囊括了 51 单片机实验所需的所有元器件，是一台功能比较完备的实验开发平台。该开发板在功能上可分为如下几个主要部分。

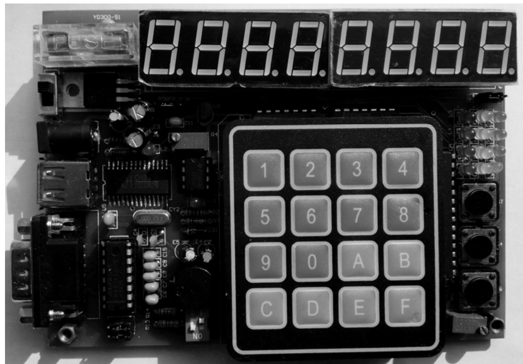


图 3-2 单片机实验开发平台图

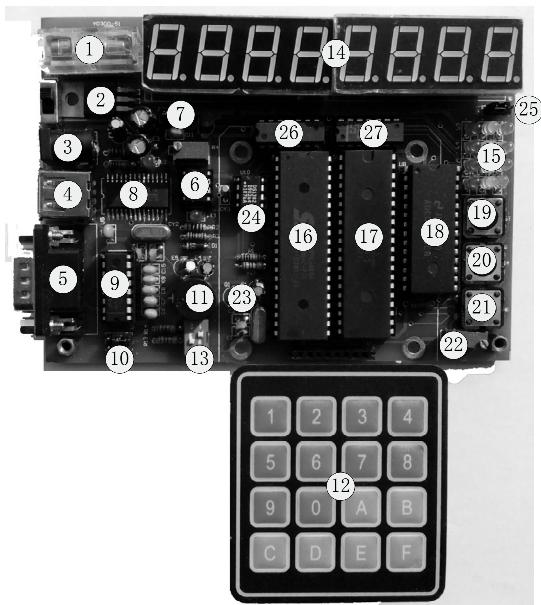


图 3-3 实验开发平台元器件图

表 3-1　实验开发平台元器件表

编　号	说　明	编　号	说　明
1	保险丝	15	发光二极管
2	LM7805	16	SST89E58RD
3	9V 开关电源接口	17	8255
4	USB 程序下载接口	18	ADC0809
5	串口	19	复位按键
6	555 电路	20	中断按键
7	7805 和 USB 供电选择跳线	21	计数器按键
8	CH341A	22	可变电阻
9	MAX3232	23	18B20
10	USB 和串口程序下载选择跳线	24	1302
11	蜂鸣器	25	P1 口复用开关
12	行列键盘	26	74LS74
13	拨码开关	27	74LS02
14	8 位 LED		

1．接口部分

与 PC 相连可通过 9 针串口，然后经过 MAX3232 芯片与 SST89E58 RD 串口相连；也可通过 USB 接口，然后经过固化专用程序的 ATMEGA8 芯片和 MAX3232 芯片与 SST89E58 RD 串口相连。串口和 USB 接口的选择要通过表 3-1 中的跳线 10 进行设置。

2．电源部分

开发板的电源可通过图 3-2 所示的 9V 开关电源经稳压器 7805 获得，也可利用 USB 接口供电，二者的选择需设置表 3-1 中的跳线 7。

3．人机交互部分

人机交互包括输入、输出两部分。输入部分由 4×4 行列键盘、微触开关、拨码开关组成；输出部分由 8 位 LED、发光二极管、蜂鸣器组成。

4．功能电路

（1）频率发生电路

555 电路为核心产生可调频率。

（2）测温电路

DS18B20 芯片作为温度传感器。

（3）时钟芯片

DS1302 作为时钟芯片。

（4）A-D 转换器

ADC0809 芯片作为 A-D 转换器。

7 号元器件所示跳线作用见表 3-2。

表 3-2 7 号元器件所示跳线作用

跳　　线	功　　能
中间针和左边相连	USB 接口供电
中间针和右边相连	7805 供电

10 号元器件所示跳线作用见表 3-3。

表 3-3 10 号元器件所示跳线作用

跳　　线	功　　能
中间针和左边相连	USB 通信下载程序
中间针和右边相连	串口通信下载程序

25 号元器件所示跳线作用见表 3-4。

表 3-4 25 号元器件所示跳线作用

跳　　线	功　　能
中间针和左边相连	P1.4～P1.7 接的发光二极管与 VCC 相连
中间针和右边相连	P1.4～P1.7 接的发光二极管不与 VCC 相连

8255 地址分配见表 3-5。

表 3-5 8255 地址分配

接 口 电 路	地 址 范 围
8255	PA：8FFFH PB：9FFFH PC：0AFFFH 控制口：0BFFFH

LED 显示码见表 3-6。

表 3-6 LED 显示码

显示数字	0	1	2	3	4	5	6	7	8	9
显示码	3FH	06H	5BH	4FH	66H	6DH	7DH	07H	7FH	6FH

LED 位选码见表 3-7。

表 3-7 LED 位选码

显示位	1	2	3	4	5	6	7	8
位选码	80H	40H	20H	10H	01H	02H	04H	80H

实验板上的串行接口是实验板与 PC 通信的基础通道，需调试的程序通过串行接口下载到实验板中。PC 上的串行接口是 RS-232，RS-232 包括了按位进行串行传输的电气和机械方

面的规定。RS-232 关于电气特性的要求规定，驱动器输出电压相对于信号地线在−5～−15V 之间为逻辑 1 电平，表示传号状态；输出电压相对于信号地线在+5～+15V 之间为逻辑 0 电平，表示空号状态。在接收端，逻辑 1 电平为−3～−15V，逻辑 0 电平为+3～+15V，即允许发送端到接收端有 2V 的电压降。这样的 RS-232 电平和 TTL 逻辑电路（单片机）产生的电平是不一样的，因此，PC 与单片机之间必须经过一定的电路转换逻辑电平。实验板上的 RS-232 串行接口逻辑电平转换电路通过 MAX3232 芯片实现。

考虑到价格和使用频率，该实验板标配时未提供液晶显示功能，但用户可根据需要来选购不同的液晶扩展板。

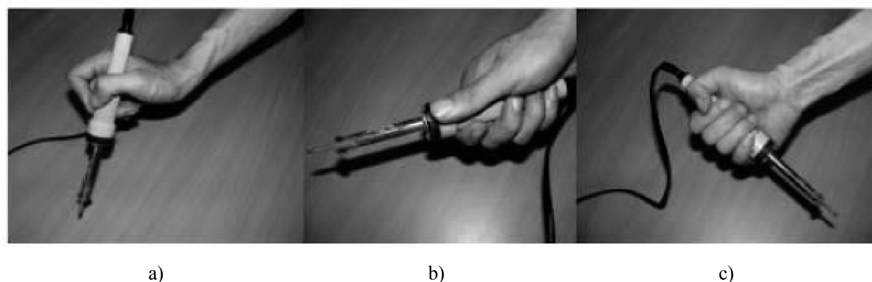
对 8 个引脚以上的集成电路均使用 IC 插座，便于用户观察和熟悉各种元器件，也便于元器件的更换和测试。

采用灵活的系统配置方式，即标配时，实验板上已提供了足够多的实验资源；对那些花费较大、耗电较多，而使用频度又较低的高端应用实验（如液晶显示、微型打印机、语音录放、步进电动机驱动、CAN 总线通信等），可通过选购和外接不同的扩展板来加以实现。

▷▷▷ 3.3.2 仿真板焊接步骤

（1）焊接的操作姿势

焊接时要注意卫生和安全，鼻子要避开焊剂的烟雾，鼻子远离烙铁 20～30cm。焊接的操作姿势如图 3-4 所示。



a) b) c)

图 3-4 焊接的操作姿势

a) 握笔法 b) 正握法 c) 反握法

（2）焊接步骤

1）准备焊接。左手拿焊锡丝，右手握烙铁。烙铁头要保持干净，这样才能粘锡。烙铁头容易被高温氧化，形成黑色杂质层。黑色杂质层隔热，所以在用前要用锉刀或砂纸打磨后镀锡。烙铁头的渣子可用湿布或湿润的海绵擦净。如需要，可以在焊件上加上焊剂，有些焊剂使焊件易于浸润焊锡，但焊剂多了焊接渣子可能过多，延长加热时间，造成浪费。有些焊剂有腐蚀性，清除工作也有些麻烦。如焊丝中已加了焊剂，则不需要在焊接时另加焊剂。

2）加热焊件。注意加热时烙铁和焊件采用面接触，不要用烙铁对焊件施加压力。对于特殊的焊接环境要采用不同的烙铁头，或自己打磨定制的烙铁头，这样能大大提高焊接工作效率。如果只有一把烙铁时就要用烙铁头上的少量锡作锡桥来传热量。加热时注意采用烙铁的加热面“面”加热，而不是“点”加热，这样导热快些。还要注意防止烙铁触在大金属体上，以至于热量被散掉。

3）送入焊丝。加热焊件达到一定温度后，焊丝从烙铁对面接触焊件。焊锡量要合适，过多的锡容易造成短路或浪费，过少的锡则不牢靠。最好不要用烙铁头送焊锡，这样会易于造成焊料的氧化，还可能导致焊料中焊剂（如松香）的挥发。如果要用烙铁头送锡，则应该快速送。另外，也可以在烙铁和焊件间的焊点处施加焊剂。

4）移开焊丝。当焊锡加热到一定量后，可以立即移开。

5）移开烙铁。当焊锡浸润焊盘或者施焊部位后，移开烙铁。注意在移开前不要抖动。

（3）焊接质量检查

电路板在焊接后需要检查，由于忽视焊接产品质量检查造成的损失屡见不鲜，若条件有限，可采用目视检查，检查外观上焊接质量是否合格，用 3～10 倍放大镜进行目视，目视检查的主要内容包括：

1）查看电路板是否有错焊、漏焊、虚焊。

2）查看是否有连焊、焊点是否拉尖的现象。

3）查看焊盘是否有脱落、焊点是否有裂纹。

4）焊点外形润湿应良好，焊点表面是否光亮、圆润。

5）焊点周围是否有残留焊剂。

6）焊接部位有无热损伤和机械损伤现象。

采用目视检查外，还可以采用手触检查。在外观检查中发现有可疑现象时，采用手触检查。主要是用手指触摸元器件看是否有松动、焊接不牢固现象，用镊子轻轻拨动焊接部位或夹住元器件连线，轻轻拉动看是否有松动现象。

▷▷▷ 3.3.3　仿真板调试步骤

一个单片机系统的可靠性是其自身软硬件与其所处工作环境综合作用的结果，因此系统的可靠性也应从这两个方面来分析与设计。对于系统自身，能否在保证系统各项功能实现的同时，对系统自身运行过程中出现的各种干扰信号及直接来自于系统外部的干扰信号进行有效的抑制，是决定系统可靠性的关键。有缺陷的系统往往只从逻辑上保证系统功能的实现，而对于系统运行过程中可能出现的潜在的问题考虑欠缺，采取的措施不足，在干扰信号真正袭来的时候，系统就可能会陷入困境。任何系统的可靠性都是相对的，在一种环境下能够很好工作的系统在另一种环境下却有可能是很不稳定的。这就充分说明环境对系统可靠运行的重要性。在针对系统运行环境设计系统的同时，应尽量采取措施改善系统运行的环境，降低环境干扰，但这样的措施往往比较有限。

提高单片机系统可靠性的方法与措施很多。一般的，应根据系统所面临的具体的可靠性问题，针对引起或影响系统不可靠的因素采取不同的处理措施。这些措施一般从这样两个目的出发：第一，尽量减少引起系统不可靠或影响系统可靠的外界因素；第二，尽量提高系统自身抗干扰能力及降低自身运行的不稳定性。例如，为了抑制电源的噪声和环境干扰信号而采用的滤波技术、隔离技术、屏蔽技术等都是出于第一个目的；另外，针对系统自身而采用的看门狗电路、软件抗干扰技术、备份技术等均是出于第二个目的而采取的措施。其中第一类措施较常使用，其使用简单而且效果也较好，但其对系统可靠性的提高是有限的，许多情况下不能满足系统的要求；第二类措施的使用可以更进一步提高系统的可靠性，往往在高可靠性的系统设计中被广泛使用。

51 单片机仿真板调试步骤如下：

（1）断电检测

所谓断电检测，就是在不通电的情况下，通过万用表和目测的方法来检查电路板的电路连通情况，要注意可能的漏焊、虚焊、短路、开路现象。对照电路原理图线路和电路板焊接线进行对比，检查线路的连通性就显得非常重要，很多初学者经常会漏接线、少接线，甚至焊错线。如果电路连接错误，当然也就无法进行下一步的电路功能检测了。

特别注意：

1）电源对地（8051 的 40 对 20 或 74XX 的 20/16/14 对 10/8/7）之间不能短路；

2）所有的电源端、地端应当分别相通，如 8051 的 40 引脚、LS373 的 20 引脚、LS138 的 16 引脚都是通的。

（2）通电检测

硬件电路调试的基本原则是将单元模块电路按照顺序依次加入并分别进行排查，检查各模块电路的功能是否正常。

模块电路的通电检测：首先芯片座里不插芯片的情况下通电，用万用表检测各点的电压，如电源电压、接地点电压、按键动作时的电压变化等是否正常。然后插入芯片后再次通电，注意观察各元器件是否有温度过热、烧焦味道等异常现象，排除异常后才能进行最终模块功能的检测。对于单片机制作来说，主要包括电源模块、单片机系统模块、外围电路模块。

1）电源模块。通常由整流稳压电路构成，只需要用万用表或者示波器测量其输出电压是否满足要求即可，通常应为+5V 的直流电压。

2）单片机系统模块。通常包括单片机芯片及其时钟电路、复位电路，以及可能的地址锁存器电路等，如闭环温控装置中用到的单片机系统模块。由于单片机是可编程芯片，需要编写简单的测试程序并下载到单片机芯片中运行，然后才能检测单片机系统模块工作是否正常。通常单片机工作必须满足 4 个基本条件：

① 两高两低。40 引脚 VCC、31 引脚 EA 必须高；9 引脚 RST、20 引脚 GND 必须低。

② 复位正常。有复位按键的，断开时，9 引脚 RST 为低，按键后，则变高；无复位按键的，开机瞬间，有一个高冲过程（指针表看很明显）。

③ 有振荡。用示波器测量 8051 的 30 引脚 ALE 是否有振荡波形输出，且其频率应为时钟振荡频率的六分之一。如晶振为 12MHz，则 ALE 信号的频率应为 2MHz。

④ 电流正常。电路板上只有 8051 或其他 CPU，没有输出 LED 数码管等耗电器件时，电源总电流一般为几毫安～十几毫安，如果电流过大有可能是 8051 的 I/O 引脚有短路现象。

运行程序检测每个 I/O 引脚波形，恒高、恒低或不高不低都可能有问题，需要进一步分析。如果单片机系统模块还有地址锁存器电路，则需要编写一段下面的测试程序下载运行。然后用示波器测量 A0～A2 引脚，随着 ALE 的变化，各引脚将输出相应波形，否则表明地址锁存器电路部分有故障。地址锁存器测试程序：

```
MOV A, #00H
LOOP: MOV DPTR, #01H
MOVX @DPTR, A
```

```
MOV DPTR, #02H
MOVX @DPTR, A
MOV DPTR, #04H
MOVX @DPTR,   A
LJMP LOOP
END
```

3）外围电路模块。不同的外围电路，具有不同的功能，需要设计不同的接口电路才能与单片机连接，检测的内容和方法也不尽相同。例如，对于闭环温度控制系统的 A-D 转换电路模块，最重要的是检查：当输入的（温度）模拟电压变化时，输出的数字量是否同步成正比地变化，编一段测试程序来启动 A-D 转换并将结果输出到某一个 I/O 口如 P1，对照模拟量检查 I/O 口上的数字量。有仿真器的可以运行到断点或单步调试，来观察模拟输入的转换结果。

▷▷ 3.4 案例笔记

▷▷▷ 3.4.1 电源符号

电子电路中，常可以看到电路中 VCC、VDD、VSS、GND 和 AGND 区别的五种不同的符号，它们有什么区别呢？

1）VCC：C=circuit 表示电路的意思，即接入电路的电压。

2）VDD：D=device 表示元器件的意思，即元器件内部的工作电压。

3）VSS：S=series 表示公共连接的意思，通常指电路公共接地端电压。

需要说明的是：

1）对于数字电路来说，VCC 是电路的供电电压，VDD 是芯片的工作电压，VSS 是接地点。

2）有些 IC 既有 VDD 引脚又有 VCC 引脚，说明这种元器件自身带有电压转换功能。

3）在场效应晶体管（或 CMOS 器件）中，VDD 为漏极，VSS 为源极，VDD 和 VSS 指的是器件引脚，而不表示供电电压。

▷▷▷ 3.4.2 接地的分类

无论是在模拟电路中还是在数字电路中都存在着各种各样的“地”，为便于大家了解和掌握，现将其总结出来，供大家参考。

1）信号“地”：信号“地”又称参考“地”，就是零电位的参考点，也是构成电路信号回路的公共段。图形符号为“⊥”。

① 直流地：直流电路“地”，零电位参考点。

② 交流地：交流供电电源的地线，这种地通常是产生噪声的地，且不是零线。

③ 功率地：大电流网络器件、功放器件的零电位参考点。

④ 模拟地：放大器、采样保持器、A-D 转换器和比较器的零电位参考点。

⑤ 数字地：也叫逻辑地，是数字电路的零电位参考点。

2）保护“地”：保护“地”是为了保护人员安全而设置的一种接线方式。保护“地”线一端接用电器，另一端与大地作可靠连接。

3）不同地线的处理方法：模拟地与数字地，虽然最终都要连接到一块的，但还是要区分模拟地和数字地。这是因为虽然是相通的，但是受距离影响，同一条导线，不同点的电压可能是不一样的，特别是电流较大时，因为导线存在着电阻，电流流过时就会产生压降；另外，导线还有分布电感，在交流信号下，分布电感的影响就会表现出来。当数字信号的高频噪声很大时，如果模拟地和数字地混合，就会把噪声传到模拟部分，造成干扰；如果分开接地，则高频噪声可以在电源处通过滤波来隔离掉，但如果两个地混合，就不好滤波了。

▷▷▷ 3.4.3　干扰基础知识

（1）干扰的基本要素

形成干扰的基本要素有三个：

1）干扰源。指产生干扰的元器件或设备，通常在电压变化率或电流变化率大的地方，如图 3-5 所示。如果在控制系统附近存在磁场、电磁场、静电场或电磁波辐射源，就可能通过空间感应，直接干扰系统中的各设备（控制器、驱动接口、转换接口等）和导线，使其中的电平发生变化，或产生脉冲干扰信号。系统附近或系统中的感性负载是最常见的干扰源，它的开、停会引起电磁场的急剧变化，其连接点的火花放电也会产生高频辐射。人体和处于浮动状态的设备都可能带有静电，甚至可能积累很高的电压。在静电场中，导体表面的不同部位会感应出不同的电荷，或导体上原有的电荷经感应而重新分配，这些都将干扰控制系统的正常运行。

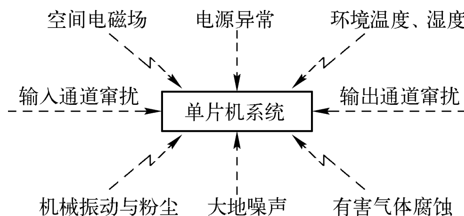


图 3-5　干扰源

2）传播路径。指干扰从干扰源传播到敏感器件的通路或媒介。典型的干扰传播路径是导线传导和空间辐射。电磁干扰就其实际作用于电路的机理有四种传输方式：传导耦合、磁场耦合、电场耦合、电磁场耦合（磁场耦合与电场耦合），如图 3-6 所示。

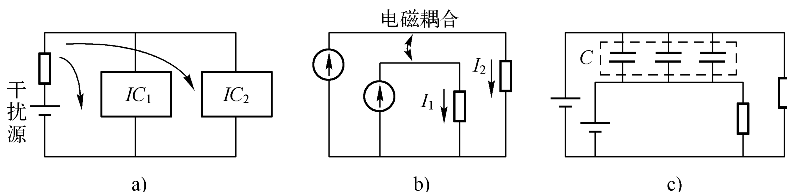


图 3-6　电磁干扰传输方式

a) 传导耦合　b) 磁场耦合　c) 电场耦合

不同传播方式干扰的传播途径及表现形式见表 3-8。

表 3-8 不同传播方式干扰的传播途径及表现形式

传播途径	传播方式	决定因素	干扰表现形式
导线	传导	经导线侵入	串模干扰：叠加于往返两线间的干扰
			共模干扰：叠加于线路和地线间的干扰
			电源线：从电源电路侵入的干扰
			信号线：从信号输入线、输出线侵入的干扰
			控制线：从控制线侵入的干扰
空间	辐射	辐射电磁场距离和辐射源的波长	电磁波
	感应	平行配线和多芯电缆等近距离电磁场	静电感应：高阻抗电场静电耦合
			电磁感应：低阻抗磁场电磁耦合
大地和接地电路	地线传导	地线上出现的干扰电压	通过静电耦合和电磁耦合，由地线侵入的干扰
			电导耦合：外电流流入裸线
			天线效应：接地线成为天线，辐射出干扰
	接地干扰	地电流	共模干扰：由接地点间电位差引起的干扰

3）敏感器件。指容易被干扰的对象，如 A-D、D-A 转换器，单片机，数字 IC，弱信号放大器等。

干扰的基本要素缺一不可，其中一个条件不存在，就不可能发生电磁兼容问题。其实，可以把干扰问题与传染病进行类比，任何传染病的爆发、流行，也正需要有这三个要素——病原体、传播途径和易感人群。消灭病原体、切断传播途径、提供易感人群的免疫力是抑制传染病的措施，而抑制干扰源、切断干扰传播路径、提高敏感器件的抗干扰性能是解决干扰问题的基本思路。

（2）干扰的分类

常见干扰的种类见表 3-9。

表 3-9 干扰的种类

分类方式	干扰的种类	
按干扰信号与有用输入信号的传输关系分类	串模干扰、共模干扰	
按干扰传播方式分类	静电干扰、磁场耦合干扰、电磁辐射干扰、共阻抗干扰、漏电耦合干扰等	
按干扰来源分类	内部干扰	过渡干扰、线间干扰、电源干扰、电弧和反电势干扰、接地系统干扰、漏磁干扰、传输线反射干扰、漏电干扰
	外部干扰	辐射干扰、电网干扰、周围用电干扰、接地干扰、传输线反射干扰、外部线间串扰
按干扰形式分类	交流干扰、直流干扰、不规则噪声干扰、机内调制干扰	
按干扰出现规律分类	固定干扰、随机干扰	

电路板内部常见电磁干扰有：

1）平行线效应，即平等导线间存在电感效应、电阻效应、电导效应、互感效应、电容效应。

2）天线效应，即一定形状的导体对一定波长的电磁波可产生发射或接收的天线效应，在高频电路中不能忽视。

3）电磁感应，即磁性元件的影响。



（3）单片机系统干扰的消除措施

要使单片机系统具有良好的抗干扰性能，印制电路板的设计十分关键。一个具有良好的电磁兼容性的印制电路板，必须按高频电路来设计。尽管单片机系统大部分电路的工作频率并不高，但是EMI（电磁干扰）的频率是高的，EMC（电磁兼容性）测试的模拟干扰频率也是高的。因此单片机的印制电路板设计必须考虑高频电路的特点：

1）要有良好的地线层。良好的地线层处处等电位，不会产生共模电阻耦合，也不会经地线形成环流产生天线效应；良好的地线层能使 EMI 以最短的路径进入地线而消失。建立良好的地线层最好的办法是采用多层板，一层专门用作地线层；如果只能用双面板，应当尽量从正面走线，反面用作地线层，不得已才从反面过线。

2）保持足够的距离。对于可能出现有害耦合或辐射的两根线或两组线要保持足够的距离，如滤波器的输入与输出、光耦的输入与输出、交流电源线与弱信号线等。

3）长线加低通滤波器。走线尽量短，不得已走的长线应当在合理的位置插入电容、*RC*或 *LC* 低通滤波器。

4）除了地线，能用细线的不要用粗线。因为 PCB 上的每一根走线既是有用信号的载体，又是接收辐射干扰的天线，走线越长、越粗，天线效应越强。

5）单片机印制电路板上不但每一根走线都存在天线效应，每一个元器件也存在天线效应，元器件的导电部分越大，天线效应越强，所以，同一型号芯片，封装尺寸小的比封装尺寸大的天线效应弱。这就解释了许多工程师已经注意到的一个现象：同一装置，采用贴片元器件比采用双列直插元器件更易通过 EMC 测试。不仅是集成电路芯片，电阻、电容封装也与 EMC 有关。贴片元器件比非贴片元器件干扰小。

此外，天线效应还与每个芯片的工作电流环路有关。要削弱天线效应，除了减小封装尺寸，还应尽量减小工作电流环路尺寸、降低工作频率和电流变化率。留意最新型号的集成电路芯片（尤其是单片机）的引脚布局会发现：它们大多抛弃了传统方式即左下角为 GND、右上角为 VCC，而将 VCC 和 CND 安排在相邻位置，就是为了减小工作电流环路尺寸。

单片机应用系统的主要干扰渠道还有空间干扰、过程通道干扰、供电系统干扰。应用于工业生产过程的单片机应用系统中，应重点防止供电系统与过程通道的干扰。除此之外，提高单片机本身的可靠性措施还包括降低外时钟频率，采用时钟监测电路与看门狗技术、低电压复位、EFT 抗干扰技术、指令设计上的软件抗干扰等几个方面。

▷▷▷ 3.4.4 如何判断集成电路的好坏

准确判断集成电路的好坏是电路调试的一个重要内容，判断不准，往往花大力气换上新集成电路而故障依然存在，所以要对集成电路作出正确判断，首先要掌握该集成电路的用途、内部结构原理、主要电特性等，必要时还要分析芯片内部原理图。除了这些，如果还有各引脚对地直流电压、波形、对地正反向直流电阻值，就为判断提供了有利依据；然后按故障现象判断其部位，再按部位查找故障元件。有时需要多种判断方法来证明该元器件是否确实损坏。

一般对集成电路的检查判断方法有两种：一是不在线判断，即集成电路未焊入印制电路板的判断。这种方法在没有专用仪器设备的情况下，要确定该集成电路的质量好坏是很困难的，一般情况下可用直流电阻法测量各引脚对应于接地脚间的正反向电阻值，并和完好集成

电路进行比较，也可以采用替换法把可疑的集成电路插到正常设备同型号集成电路的位置上来确定其好坏。当然有条件可利用集成电路测试仪对主要参数进行定量检验，这样使用就更有保证。二是在线检查判断，即集成电路连接在印制电路板上的判断方法。在线检查判断是检修集成电路在电视、音响、录像设备中最实用的方法。以下分几种情况进行阐述。



（1）电压测量法

主要是测出各引脚对地的直流工作电压值，然后与标称值进行比较，依此来判断集成电路的好坏。用电压测量法来判断集成电路的好坏是检修中最常采用的方法之一，但要注意区别非故障性的电压误差。测量集成电路各引脚的直流工作电压时，如遇到个别引脚的电压与原理图或维修技术资料中所标电压值不符，不要急于断定集成电路已损坏，应该先排除以下几个因素后再确定。

1）所提供的标称电压是否可靠，因为有一些说明书、原理图等资料上所标的数值与实际电压有较大差别，有时甚至是错误的。此时，应多找一些有关资料进行对照，必要时分析内部原理图与外围电路再进行理论上的计算或估算来证明电压是否有误。

2）要区别所提供的标称电压的性质，其电压是属哪种工作状态的电压。因为集成电路的个别引脚随着注入信号的不同而明显变化，所以此时可改变输入信号，再观察电压是否正常。如后者为正常，则说明标称电压属正常工作电压，而这工作电压又是针对某一特定的条件下而言的，即测试的工作状态不同，所测电压也不一样。

3）要注意由于外围电路可变元器件引起的引脚电压变化。当测量出的电压与标称电压不符时可能因为个别引脚或与该引脚相关的外围电路，连接的是一个阻值可变的电位器或者是开关。这些电位器和开关所处的位置不同，引脚电压会有明显不同，所以当出现某一引脚电压不符时，要考虑引脚或与该引脚相关联的电位器和开关的位置变化，可旋动触头看引脚电压能否在标称值附近。

4）要防止由于测量造成的误差。万用表表头的内阻不同或不同的直流电压档都会造成测量误差。一般原理上所标的直流电压都是以测试仪表的内阻大于 20kΩ/V 进行测试的。内阻小于 20kΩ/V 的万用表进行测试时，将会使被测结果低于原来所标的电压。另外，还应注意不同电压档上所测的电压会有差别，尤其用大量程档时，读数偏差影响更显著。

5）当测得某一引脚电压与正常值不符时，应根据该引脚电压对 IC 正常工作有无重要影响以及其他引脚电压的相应变化进行分析，才能判断 IC 的好坏。

6）若 IC 各引脚电压正常，则一般认为 IC 正常；若 IC 部分引脚电压异常，则应从偏离正常值最大处入手，检查外围元器件有无故障，若无故障，则 IC 很可能损坏。

7）对于动态接收装置，在有无信号时，IC 各引脚电压是不同的。如发现引脚电压不该变化的反而变化大，该随信号大小和可调元器件不同位置而变化的反而不变化，就可确定 IC 损坏。

8）对于多种工作方式的装置，在不同工作方式下，IC 各引脚电压也是不同的。

以上几点就是在集成电路没有故障的情况下，由于某种原因而使所测结果与标称值不同，所以总的来说，在进行集成电路直流电压或直流电阻测试时要规定一个测试条件，尤其是作为实测数据记录时更要注意这一点。通常把各电位器旋到机械中间位置，信号源采用一定场强下的标准信号，当然，如能再记录各功能开关位置，那就更有代表性。如果排除以上几个因素后，所测的个别引脚电压还是不符标称值时，需进一步分析原因，但不外乎两种可

能：一是集成电路本身故障引起；二是集成电路外围电路造成。分辨出这两种故障源，也是测试集成电路好坏的关键。

除了直流电压测量法外，还可以采用交流工作电压测量法：为了掌握 IC 交流信号的变化情况，可以用带有 dB 插孔的万用表对 IC 的交流工作电压进行近似测量。检测时万用表置于交流电压档，正表笔插入 dB 插孔；对于无 dB 插孔的万用表，需要在正表笔串接一只 0.1～0.5μF 隔直电容。该方法适用于工作频率比较低的 IC。由于这些电路的固有频率不同，波形不同，所以所测的数据是近似值，或者作为有无信号的鉴别。

（2）在线直流电阻普测法

这一方法是在发现引脚电压异常后，通过测试集成电路的外围元器件好坏来判定集成电路是否损坏。由于是断电情况下测定阻值，所以比较安全，并可以在没有资料和数据而且不必了解其工作原理的情况下，对集成电路的外围电路进行在线检查，在相关的外围电路中，以快速的方法对外围元器件进行一次测量，以确定是否存在较明显的故障。具体操作是先用万用表 R×10Ω 档分别测量二极管和晶体管的正反向电阻值。此时由于电阻档位选得很低，外电路对测量数据的影响较小，可很明显地看出二极管、晶体管的正反向电阻，尤其是 PN 结的正向电阻增大或短路更容易发现。其次可对电感是否开路进行普测，正常时电感两端阻值较大，那么即可断定电感开路。继而根据外围电路元器件参数的不同，采用不同的电阻档位测量电容和电阻，检查是否是短路和开路性故障，从而排除由于外围电路引起个别引脚的电压变化。

（3）电流流向跟踪电压测量法

此方法是根据集成电路内部的外围元件所构成的电路，并参考供电电压，即主要测试点的已知电压进行各点电位的计算或估算，然后对照所测电压是否符合，来判断集成电路的好坏，该方法必须具备完整的集成电路内部电路图和外围电路原理图。

（4）在线直流电阻测量对比法

此方法是利用万用表测量集成电路各引脚对地正反向直流电阻值，并与正常数据进行对照来判断好坏。这一方法需要积累同一型号集成电路的正常可靠数据，以便和待查数据相对比。

测量时要注意以下几点：

1）测量前要先断开电源，以免测试时损坏电表和元器件。

2）万用表电阻档的内部电压不得大于 6V，量程最好用 R×100 或 R×1k 档。

3）测量 IC 引脚参数时，要注意测量条件，如被测机型、与 IC 相关的电位器的滑动臂位置等，还要考虑外围电路元器件的好坏。

（5）非在线数据与在线数据对比法

所谓非在线数据是指集成电路未与外围电路连接时，所测得的各引脚对应于地的正反向电阻值。非在线数据通用性强，可以对不同机型、不同电路、集成电路型号相同的电路进行对比。具体测量对比方法如下：首先应用空心针头和铬铁使被查集成电路的接地脚与印制电路板脱离，再对应于某一怀疑引脚进行测量对比。如果被怀疑引脚有较小阻值电阻连接于地或电源之间，为了不影响被测数据，该引脚也可与印制电路板开路。直至外电路的阻值不影响被测集成电路的电阻值为止。但要注意一点，直流电阻测量对比法对于同一型号的集成电路，有一定的误差和差异，对这种情况，要在了解内部结构的基础上，进行分析和判断。

（6）替换法

用替换法判断集成电路的好坏效率高，可以减少许多检查分析的麻烦。但必须注意如下几点：



1）尽量选用同型号的集成电路或可以直接代换的其他型号，这样可不改变原系统电路的引线，简便易行，容易恢复原系统的性能指标。

2）更换拆焊原系统的集成电路时，不要急躁，不能乱拔、乱撬引脚，用所具备的条件选择最适合拆卸集成电路的方法。

3）在还没有判断外围电路是否有故障，以及未经确认原集成电路已损坏之前，不要轻易替换集成电路，否则换上去的集成电路有可能再次损坏。

4）有些集成电路，虽然其型号相同，但还要考虑其型号后缀不同。例如 M5115P 与 M5115RP，二者引脚功能排列顺序相反。

5）有时采用试探性替换，此时最好先装一个专用集成电路插座，或用细导线临时连接，这样好坏对比方便。另外，在通电前电源 VCC 回路里最好再串接一直流电流表，降压电阻阻值由大到小观察集成电路总电流的变化是否正常。对于功放电路一定要按规定装好散热片。

6）在选用同功能但不同型号和不同引脚排列的集成电路代换时，还应注意以下几点：

① 尽量选用功能、引脚、电特性相近的集成电路。

② 改变引脚连线时，应尽量利用印制电路板上的孔位和线路，连线要整齐，信号线的前后段不要交叉，以免电路产生自激。

③ 集成电路的供电电压应与集成电路的电源电压 VCC 的典型值相符。

④ 集成电路的各信号输入、输出阻抗要与原电路相匹配，连接好的集成电路在通电前应作最后一次的检查，确认电路无误后再接通电源。

（7）总电流测量法

该方法是通过检测 IC 电源进线的总电流，来判断 IC 好坏的一种方法。由于 IC 内部绝大多数为直接耦合，IC 损坏时（如某一个 PN 结击穿或开路）会引起后级饱和与截止，使总电流发生变化，所以通过测量总电流的方法可以判断 IC 的好坏。也可用测量电源通路中电阻的电压降，用欧姆定律计算出总电流值。

▷▷▷ 3.4.5 电气设备维修的十项原则

1）先动口再动手。对于有故障的电气设备，不应急于动手，应先询问产生故障的前后经过及故障现象。对于生疏的设备，还应先熟悉电路原理和结构特点，遵守相应规则。拆卸前要充分熟悉每个电气部件的功能、位置、连接方式以及与周围其他元器件的关系，在没有组装图的情况下，应一边拆卸，一边画草图，并做好标记。

2）先外部后内部。应先检查设备有无明显裂痕、缺损，了解其维修史、使用年限等，然后再对机内进行检查。拆前应先排除周边的故障因素，确定为机内故障后才能拆卸，否则，盲目拆卸，可能将设备越修越坏。

3）先机械后电气。只有在确定机械零件无故障后，再进行电气方面的检查。检查电路故障时，应利用检测仪器寻找故障部位，确认无接触不良故障后，再有针对性地查看线路与机械的运作关系，以免误判。

4）先静态后动态。在设备未通电时，判断电气设备按钮、接触器、热继电器以及保险

丝的好坏，从而判定故障的所在。通电试验，听其声、测参数、判断故障，最后进行维修。如在电动机缺相时，若测量三相相间电压值无法判别时，就应该单独测每相对地电压，方可判断哪一相缺损。

5）先清洁后维修。对污染较重的电气设备，先对其按钮、接线点、接触点进行清洁，检查外部控制键是否失灵。许多故障都是由脏污及导电灰尘引起的，一经清洁故障往往会排除。

6）先电源后设备。电源部分的故障率在整个故障设备中占的比例很高，所以先检修电源往往可以事半功倍。

7）先普遍后特殊。因装配配件质量或其他设备故障而引起的故障，一般占常见故障的50%左右。电气设备的特殊故障多为软故障，要靠经验和仪表来测量和维修。

8）先外围后内部。先不要急于更换损坏的电气部件，在确认外设电路正常时，再考虑更换损坏的电气部件。

9）先直流后交流。检修时，必须先检查直流回路静态工作点，再交流回路动态工作点。

10）先故障后调试。

▷▷▷ 3.4.6　0Ω电阻

经常可以在电路中见到 0Ω的电阻，对于初学者来说，往往会很迷惑：既然是 0Ω的电阻，那就是导线，为何要装上它呢？其实 0Ω的电阻大致有以下几个功能：

1）作为跳线使用。这样既美观，安装也方便。

2）在数字和模拟等混合电路中，往往要求两个地分开，并且单点连接。可以用一个 0Ω的电阻来连接这两个地，而不是直接连在一起。这样做的好处就是，地线被分成两个网络，在大面积铺铜等处理时，就会方便得多。附带提示一下，这样的场合，有时也会用电感或者磁珠等来连接。

3）作保险丝用。由于 PCB 上走线的熔断电流较大，如果发生短路过流等故障时，很难熔断，可能会带来更大的事故。由于 0Ω电阻电流承受能力比较弱（其实 0Ω电阻也是有一定的电阻的，只是很小而已），过流时就先将 0Ω电阻熔断了，从而将电路断开，防止更大事故的发生。有时也会用一些阻值为零点几欧或者几欧的小电阻来作保险丝，不过不推荐使用。

4）为调试预留的位置。可以根据需要，决定是否安装，或者其他的值。有时也会用*来标注，表示由调试时决定。

5）作为配置电路使用。这个作用与跳线或者拨码开关类似，但却是通过焊接固定上去的，这样避免了普通用户随意修改配置。通过安装不同位置的电阻，就可以更改电路的功能或者设置地址。0Ω的电阻有不同的规格，一般是按功率来分，如 1/8W、1/4W 等。用户需要根据产品的数据手册进行选择。

案例4



指令系统的学习



4.1 案例任务

任务 1：用 5 种方式将片内 RAM 的 15H 单元内容 AAH 送 55H 单元，并利用 Keil 的存储器窗口查看中间过程及结果。

任务 2：利用堆栈操作指令将 R2 和 A 中的内容交换，并利用 Keil 的存储器窗口查看中间过程及结果。

任务 3：用 2 种查表法获得 07H 的 ASCII 码，并将其送入 21H 单元，并利用 Keil 的存储器窗口查看中间过程及结果。

任务 4：利用加法指令完成计算任务，并利用 Keil 的存储器窗口查看中间过程及结果。

4.2 案例要点

（1）Keil C 软件的使用方法

1）Keil C 软件的安装及开发环境的设置；

2）编译、连接的方法；

3）调试命令、在线汇编及断点设置；

4）利用输出窗口（Output Windows）、观察窗口（Watch&Call Stack Windows）、存储器窗口（Memory Window）、反汇编窗口（Dissambly Window）、串行窗口（Serial Window）等调试工具进行调试；

（2）指令的基本概念

1）机器指令：用二进制数 0、1 表示的命令代码，常以十六进制表示。

2）机器语言：由机器指令描述的程序语言。

3）助记符指令：用字母和十六进制数代替机器指令形成的符号指令。

4）助记符语言：由助记符指令形成的程序语言，又称汇编语言。

（3）MCS-51 单片机汇编语言指令格式

指令格式如下：

[标号：] 操作码 [操作数（目的操作数，源操作数）] [；注释]

其中：

1）标号：称符号地址，代表该指令第 1 字节所在的地址，可有可无。

2）操作码：规定了指令将要干什么，必不可少。

3）操作数：表示参与运算的数或数的地址，可有可无。

4）注释：必须以“;”开始，可有可无。

（4）指令组成 4 要素和功能 3 要素

1）组成 4 要素：标号、操作码、操作数、注释。

2）功能 3 要素：机器码、字节数、机器周期。

（5）操作数

操作数按硬件物理单元地址可分为立即数、寄存器、存储单元 RAM（寄存器除外）/ROM、位、I/O 接口。

（6）数据传送指令

1）传送指令数量繁多。这是由于其复杂的存储器结构造成的。存储器主要分为内 RAM、ROM 和外 RAM、ROM，如图 4-1 所示。要想学好这部分内容，就要将软件和硬件结合起来理解。

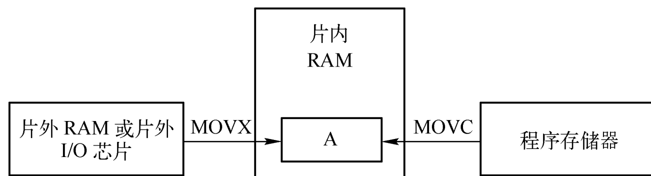


图 4-1　存储单元的传送指令

① 程序运行时内/外 ROM 同一地址单元只有一个起作用，因此内部程序存储器和外部程序存储器传送指令不像内部数据存储器和外部数据存储器分开用 MOV 和 MOVX 传送，而统一用 MOVC 指令传送。

② 由于数据存储器的同一地址能同时指向内部或外部两个 RAM 单元，因此内部 RAM 和外部 RAM 的传送指令不同，分别为 MOV 和 MOVX。

③ 片外 RAM 或 I/O 芯片、程序存储器要和片内 RAM 之间传递数据必须通过 A。

④ 片外 RAM 单元与片外 RAM 单元之间、地址不同的两个程序存储器单元之间、片外 RAM 单元与程序存储器单元之间，不能直接交换数据。

⑤ 数据传送指令不影响标志位 C、AC 和 OV。POP PSW 或 MOV PSW,#data 可能使某些标志位发生变化。

⑥ 源操作数寻址方式可分为寄存器寻址、寄存器间址、直接寻址、立即寻址和寄存器变址。目的操作数寻址方式可分为寄存器寻址、寄存器间址和直接寻址。

2）传送指令中的内部存储器单元如图 4-2 所示。由图 4-2 可见，以 A、direct、@Ri、Rn、#data 为顶点组成了五边形边边俱在，而由它们组成的内五角星形缺少 Rn 和@Ri 之间的连接，因此：

① 除 Rn 和@Ri 外，内部单元之间可以传送数据。

② 仅#data 为单向传送数据。

③ 仅 direct 能自身传递数据。

④ #data 指向的数据和 direct 指向的地址在程序运行中不能改变。而 Rn 和@Ri 在程序运行中通过指令能够改变，因此它们的使用非常灵活，在某些程序（如数据串操作）中应用

较多。

⑤ 仅 DPTR 是 16 位的，其余都是 8 位的。DPTR 是对外部存储器访问时的指针（注意体会软件依赖硬件，DPTR 是由硬件存储器寻址需要而产生的）。PC 也是 16 位的，但不能用 MOV 指令访问。

3）由图 4-2 可以看出单片机内部 RAM 非常复杂。结构复杂必使使用简单。#data、direct、Rn、@Ri 身兼不同属性，见表 4-1。程序设计中数据块在存储器中的移动非常频繁，因此数据和地址在程序运行中具有可变的属性显得非常重要，否则循环程序实现的数据块传递将不得不用固定地址和数据的指令重复数十遍才能完成。

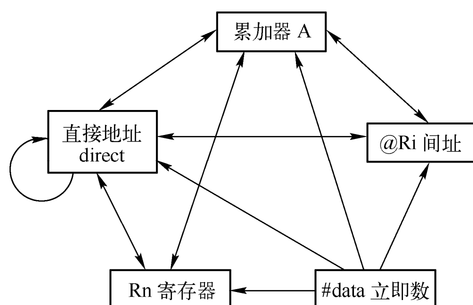


图 4-2 传送指令中的内部存储器单元

表 4-1 寻址方式总结

	固 定	可 变
数据	#data	Rn
地址	direct	@Ri

（7）算术运算类指令

注意算术运算类指令对 PSW 的影响。

（8）逻辑操作与移位指令

对于逻辑操作与移位指令，如果 direct 是 51 系列单片机的某个并行口 P0～P3（这里的端口是指并行口中具有端口地址的锁存器或寄存器，下同），则指令将以“读→修改→写”模式完成逻辑运算。

（9）控制转移指令

SJMP、AJMP、LJMP 指令的功能基本相同，但其长度却不相同，见表 4-2。同样，ACALL 和 LCALL 指令的功能基本相同，但其长度也不相同，见表 4-3。也许读者会想：有 LJMP、LCALL 这两条指令不就足够了吗？这很可能是计算机发展初期速度不够快，内存很贵，因此程序员编程时尽量选用字节数较少、实时性高的指令。如今，计算机的速度越来越快，尽管上述指令功能有些“重叠”，但它们像“化石”一样，仍保留在计算机语言中，也算是计算机硬件发展的历史痕迹吧！

表 4-2 SJMP、AJMP、LJMP 长度和周期的比较

指 令	长 度	周 期
SJMP	2	2
AJMP	2	2
LJMP	3	2

表 4-3 ACALL、LCALL 长度和周期的比较

指 令	长 度	周 期
ACALL	2	2
LCALL	3	2

（10）位操作指令

51 系列单片机中特有布尔处理器，通过 51 系列单片机的位操作指令进行逻辑设计，可



把逻辑表达式直接变换成软件执行，方法简便，免去了过多的数据往返传送、指令屏蔽，大大简化了编程，节省存储器空间，加快了处理速度，还可实现复杂的组合逻辑处理功能。所有这些，特别适合于某些数据采集、实时测控等应用系统。

4.3 案例设计

（1）编写完成任务 1 的汇编程序

1）解法 1。汇编程序如下：

```
ORG    0000H
MOV    15H,#0AAH
MOV    55H,15H
END
```

单击单步执行，第 1 条指令执行后，打开存储器显示窗口，在 Address 框中输入 d:15H 显示内部 RAM 的 15H 单元内容，如图 4-3 所示，说明第 1 条传送指令将立即数 AAH 送到了内部 RAM 的 15H 单元。执行第 2 条指令后，如图 4-4 所示，内部 RAM 的 15H 单元的内容传送到 55H 单元。

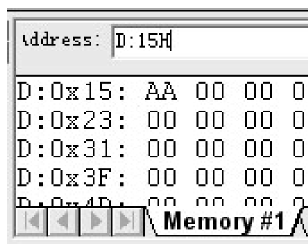


图 4-3　解法 1 存储器窗口显示 1

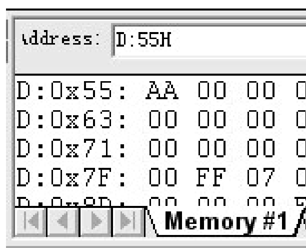


图 4-4　解法 1 存储器窗口显示 2

2）解法 2。汇编程序如下：

```
ORG    0000H
MOV    15H,#0AAH
MOV    R6,15H
MOV    55H,R6
END
```

第 2 条指令执行后，如图 4-5 所示，内部 RAM 中 15H 单元的内容已送入 R6 中；继续执行第 3 条指令，结果显示与图 4-4 相同。

3）解法 3。汇编程序如下：

```
ORG    0000H
MOV    15H,#0AAH
MOV    R1,#15H
MOV    55H, @R1
END
```

第 2 条指令执行后，如图 4-6 所示，立即数 15H 单元的内容已送入 R1 中；继续执行第 3 条指令，对 R1 进行间接寻址即将 R0 中保存的 15H 看成地址，将 15H 单元的内容送到 55H 单元，结果显示与图 4-4 相同。

Register	Value
Regs	
r0	0x00
r1	0x00
r2	0x00
r3	0x00
r4	0x00
r5	0x00
r6	0xaa
r7	0x00

图 4-5　解法 2 存储器窗口显示

Register	Value
Regs	
r0	0x00
r1	0x15
r2	0x00
r3	0x00
r4	0x00
r5	0x00
r6	0x00
r7	0x00

图 4-6　解法 3 存储器窗口显示

4）解法 4。汇编程序如下：

```
ORG     0000H
MOV     15H,#0AAH
MOV     A,15H
MOV     55H, A
END
```

第 2 条指令执行后，如图 4-7 所示，立即数 15H 单元的内容已送入 A 中；继续执行第 3 条指令，将 A 中的数送入 55H 单元，结果显示与图 4-4 相同。

5）解法 5。汇编程序如下：

```
ORG     0000H
MOV     15H,#0AAH
SETB    RS1
CLR     RS0
MOV     55H, R5
END
```

第 3 条指令执行后，如图 4-8 所示，PSW 中的 RS1、RS0 值改变，此时 R5 指向内部 RAM 中 15H 单元，因此 R5 中存 AAH；继续执行第 3 条指令，将 R5 中的数送入 55H 单元，结果显示与图 4-4 相同。

（2）编写完成任务 2 的汇编程序

汇编程序如下：

```
ORG     0000H
MOV     SP,#30H
MOV     R2,#05H
MOV     A,#01H
PUSH    ACC
PUSH    02H
POP     ACC
POP     02H
END
```

```
Sys
  a          0xaa
  b          0x00
  sp         0x07
   sp_max    0x07
  dptr       0x0000
  PC  $      C:0x0005
  states     3
  sec        0.00000150
  psw        0x00
```

图 4-7　解法 4 存储器窗口显示

```
Register     Value
Regs
  r0         0x00
  r1         0x00
  r2         0x00
  r3         0x00
  r4         0x00
  r5         0xaa
  r6         0x00
  r7         0x00
Sys
  a          0x00
  b          0x00
  sp         0x07
   sp_max    0x07
  dptr       0x0000
  PC  $      C:0x0005
  states     3
  sec        0.00000150
  psw        0x10
```

图 4-8　解法 5 存储器窗口显示

第 3 条指令执行后，SP 寄存器的值如图 4-9 所示，为 30H；

第 4 条指令执行后，结果如图 4-10 所示，SP 寄存器的值变成 31H，内部 RAM 中 31H 的单元变成了 A 中保存的 01H，注意体会 PUSH 指令“先增 1，再存储”的特点；

```
Register     Value
Regs
  r0         0x00
  r1         0x00
  r2         0x05
  r3         0x00
  r4         0x00
  r5         0x00
  r6         0x00
  r7         0x00
Sys
  a          0x01
  b          0x00
  sp         0x30
   sp_max    0x30
  dptr       0x0000
  PC  $      C:0x0007
  states     4
  sec        0.00000200
  psw        0x01
```

图 4-9　任务 2 存储器窗口显示 1

```
Sys
  a          0x01
  b          0x00
  sp         0x31
   sp_max    0x31
  dptr       0x0000
  PC  $      C:0x0009
  states     6
  sec        0.00000300
  psw        0x01
```

a)

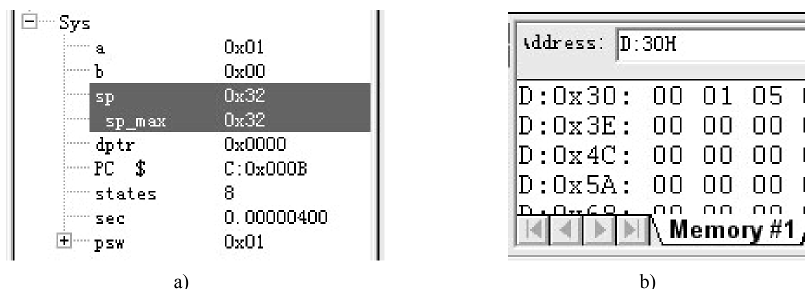
```
Address: D:30H
D:0x30:  00 01 00 0
D:0x3E:  00 00 00 0
D:0x4C:  00 00 00 0
D:0x5A:  00 00 00 0
Memory #1
```

b)

图 4-10　任务 2 存储器窗口显示 2

第 5 条指令执行后，结果如图 4-11 所示，SP 寄存器的值变成 32H，内部 RAM 中 32H 的单元变成了 R2 中保存的 05H；

第 6 条指令执行后，结果如图 4-12 所示，SP 寄存器的值变成 31H，A 中保存的值变成 05H，注意体会 POP 指令“先弹栈，再减 1”的特点；



a)　　　　b)

图 4-11　任务 2 存储器窗口显示 3

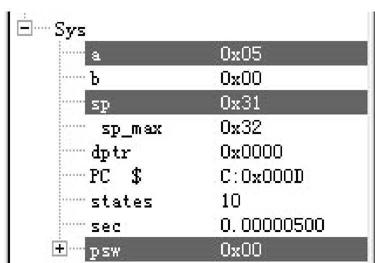


图 4-12　任务 2 存储器窗口显示 4

第 7 条指令执行后，结果如图 4-13 所示，SP 寄存器的值变成 30H，R2 中保存的值变成 01H，完成了 R2 和 A 的交换，注意内部 RAM 存储单元中 31H、32H 仍然保留 01H、05H 没有变。

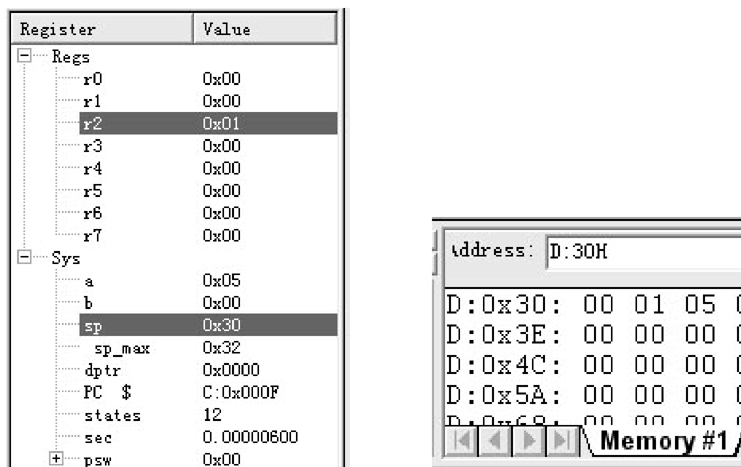


图 4-13　任务 2 存储器窗口显示 5

上述程序需要说明的是：

1）由于 PUSH 指令不能寄存器寻址，即 R2 不能压栈弹栈，因此压栈弹栈操作需对直接地址 02H 操作。寄存器指令不能压栈的原因可能是受 RS0、RS1 影响，使 Rn 物理位置不固定，压栈操作后，若物理位置改变，会给程序调试带来麻烦，并且错误非常隐蔽，不易发现。



因此计算机工程师经过编程实践，决定寄存器不能压栈。而特殊功能寄存器虽名为寄存器，但寻址方式属直接寻址，因此可以进行堆栈操作。读者可利用 Keil 调试软件进行实验验证：

```
PUSH    A            ;编译通过
PUSH    E0H          ;编译通过
PUSH    R0           ;编译不通过
PUSH    00H          ;编译通过
```

2）若入栈和出栈的顺序相反，可完成两数交换。

（3）编写完成任务 3 的汇编程序

1）解法 1。使用 MOVC A,@A+PC 指令编程，汇编程序如下：

```
            ORG     0000H
            ACALL   BCD_ASCII
            ORG     0030H
BCD_ASCII:  MOV     A, #07H          ; A＝07H
            ADD     A,#3             ;A＝A+03H=0AH，修正偏移量
            MOVC    A,@A+PC          ;
            MOV     21H,A
            RET
TAB:        DB      30H
            DB      31H
            DB      32H
            DB      33H
            DB      34H
            DB      35H
            DB      36H
            DB      37H
            DB      38H
            DB      39H
```

程序开始运行前可以用 C:30H 查看一下 ROM 中存储内容，如图 4-14 所示，30H～37H 存储的是汇编程序对应的机器码，37H 之后为 TAB 表格。执行 BCD_ASCII 子程序中的 MOVC A, @A+PC 指令后，结果如图 4-15 所示，07H 对应的 ASCII 结果已保存在 A 中。

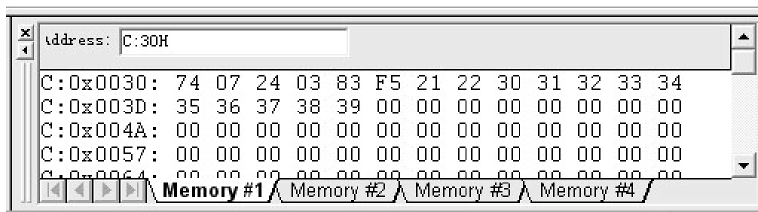


图 4-14　任务 3 存储器窗口显示 1

2）解法 2。使用 MOVC A, @A+DPTR 指令编程，汇编程序如下：

```
BCD_ASCII:  MOV     A,20H            ;A＝07H
            ADD     DPTR,#TAB
```

```
        MOVC    A,@A+DPTR
        MOV     21H,A
        RET
TAB:    DB      30H
        （以下存储单元略）
```

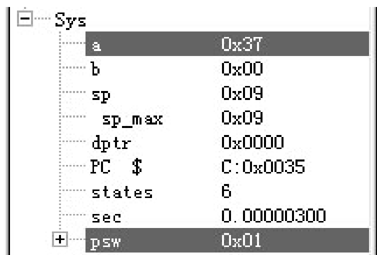


图 4-15　任务 3 存储器窗口显示 2

对于程序存储器数据传送指令需要说明的是，DPTR 可以通过传送指令 MOV 改变，而 PC 不能通过传送指令改变。PC 和 DPTR 的区别可从任务 3 的解法 1 和解法 2 体现出来。

（4）编写完成任务 4 的汇编程序

编写完成 64H+64H 的汇编程序，并观察 Keil 的寄存器窗口中 PSW 值是否正确。

分析：64H 的无符号数和有符号数表示见表 4-4。

表 4-4　64H 所表示的无符号数和有符号数

数　制	十六进制	二　进　制	十　进　制	
			无符号数	有符号数
数值	64H	0110 0100B	100	+100

64H+64H 的运算过程如图 4-16 所示。

因此 64H+64H 后，CY 和 OV 结果见表 4-5。

表 4-5　64H+64H 的 CY 和 OV 结果

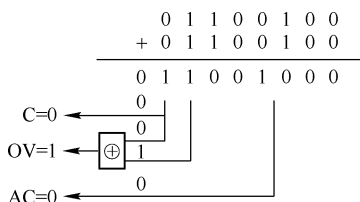
符　号	现　象	本　质
CY	C=CP=0	100+100=200<255
OV	CP ⊕ CS=1	（+100）+（+100）=+200>+127

按照 PSW 定义，由图 4-16 可知运算后 PSW 为 0000 0101B。

完成加法任务的汇编程序如下：

```
ORG     0000H
MOV     A,#64H
ADD     A,#64H
END
```

第 1 条指令执行后，结果如图 4-17 所示；第 2 条指令执行后，结果如图 4-18 所示。



Sys	
a	0x64
b	0x00
sp	0x07
sp_max	0x07
dptr	0x0000
PC $	C:0x0002
states	1
sec	0.00000050
psw	0x01

Sys	
a	0xc8
b	0x00
sp	0x07
sp_max	0x07
dptr	0x0000
PC $	C:0x0004
states	2
sec	0.00000100
psw	0x05

图 4-16　64H+64H 的运算过程　图 4-17　任务 4 存储器窗口显示 1　图 4-18　任务 4 存储器窗口显示 2

继续完成如下加法程序，并分析 PSW 的值：

1）ABH+FFH。

2）ABH+FFH。

3）9AH +E3H。

4.4 案例笔记

4.4.1 Keil C 的辅助调试功能

单片机的程序调试分为两种，一种是使用软件模拟调试，第二种是硬件调试。使用软件模拟调试，就是用计算机来模拟单片机的指令执行，并虚拟单片机片内资源，从而实现调试的目的。但是软件调试存在一些问题，不可能像真正的单片机运行环境那样，执行的指令能在同一个时间完成（往往比单片机慢）。软件调试只能是一种初步的、小型工程的调试。

Keil 软件在调试程序时提供了多个窗口，主要包括输出窗口、观察窗口、存储器窗口、反汇编窗口和串行窗口等。进入调试模式后，可以通过菜单 View 下的相应命令打开或关闭这些窗口，如图 4-19 所示。

下面具体讨论相关子窗口的功能。

（1）左侧的工程寄存器窗口

图 4-20 是工程寄存器窗口（Project Workspace）。寄存器页包括当前的工作寄存器组和系统寄存器组，系统寄存器组有一些是实际存在的寄存器，如 A、B、DPTR、SP、PSW 等，有一些是实际中并不存在或虽然存在但不能对其操作的，如 PC 等。Regs 是片内内存的相关情况值；Sys 是系统一些累加器、计数器等。

根据指令执行的不同，上述值会有相应的变化，程序员可以监测这些在单片机中看不到的值而达到调试的目的。

（2）存储器窗口

存储器窗口如图 4-21 所示。存储器窗口中可以显示系统中各种内存的值，通过在 Address 后的文本框内输入“字母:数字”即可显示相应内存值，其中字母可以是 C、D、I、X，分别代表代码存储空间、直接寻址的片内存储空间、间接寻址的片内存储空间、扩展的外部 RAM 空间，数字代表想要查看的地址。例如输入 D:0 即可观察到地址 0 开始的片内 RAM 单元值，键入 C:0 即可显示从 0 开始的 ROM 单元中的值，即查看程序的二进制代码。该窗口的显示值可以以各种形式显示，如十进制、十六进制、字符型等，改变显示方式

的方法是单击鼠标右键，在弹出的快捷菜单中选择。

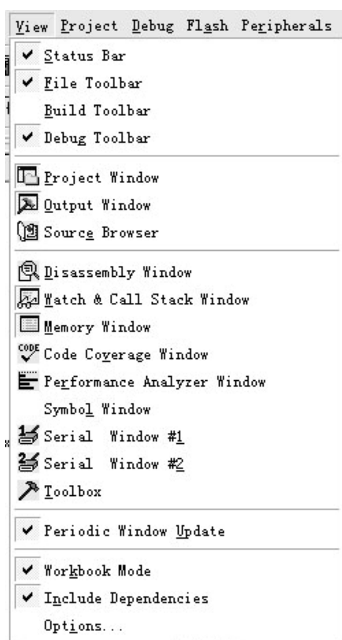


图 4-19　Keil 软件的 View 菜单

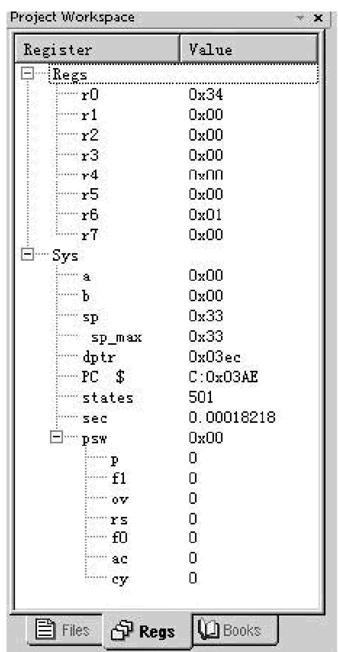


图 4-20　工程寄存器窗口

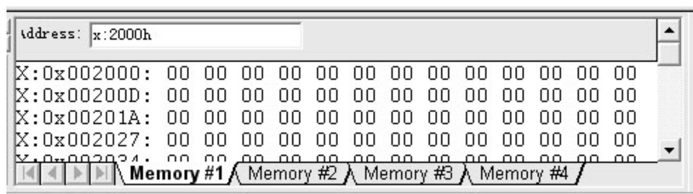


图 4-21　存储器窗口

（3）外设窗口

1）I/O 口窗口。I/O 口窗口如图 4-22 和图 4-23 所示，Port0～Port3 就对应于单片机的四个 P0～P3 口，共 32 个引脚。

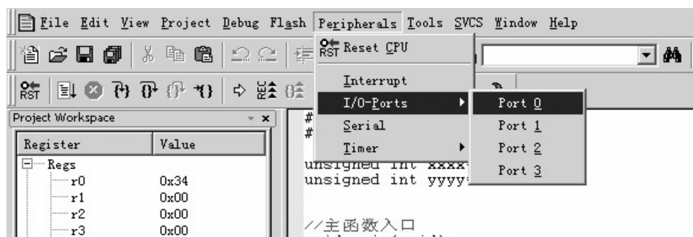


图 4-22　I/O 口菜单项

虽然软件调试无法实现硬件调试那样的信号输出，但是软件调试也可以在软件窗口监测



输出信号的高低电平，以及单片机相关端口的变化，实现模拟监测输出信号的目的。

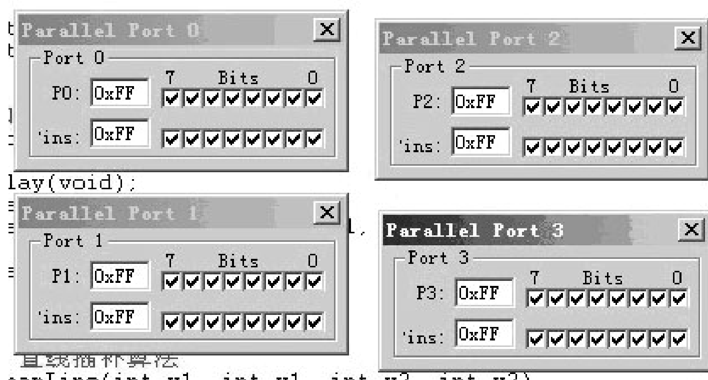


图 4-23　I/O 口窗口

2）中断窗口。中断菜单项如图 4-24 所示。

选择图 4-24 中菜单命令“Peripherals”→“Interrupt”可以打开输入预设窗口，输入值窗口如图 4-25 所示，选择不同的 Int Source 会有不同的 Selected Interrupt 的变化，通过选择与赋值达到模拟输入的目的。



图 4-24　中断菜单项

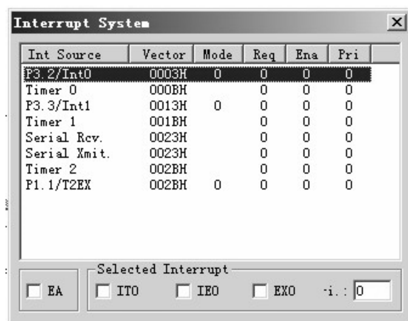


图 4-25　中断窗口

3）串口窗口。串口菜单项如图 4-26 所示，串口窗口如图 4-27 所示。



图 4-26　串口菜单项



图 4-27　串口窗口

监测串口数据的还有一个窗口，如图 4-28 所示，可单击工具栏中的出现，这个窗口可以监测从串口输出的 ASCII 代码。



图 4-28 串口监测窗口

4）定时器窗口。定时器菜单项如图 4-29 所示。定时器的设置窗口如图 4-30 所示。3 个定时器与 1 个看门狗，设置定时器的数量与工程选择的单片机型号有关系，如果是 8051 就只有 2 个定时器，如果选择 8052 就有 3 个定时器。

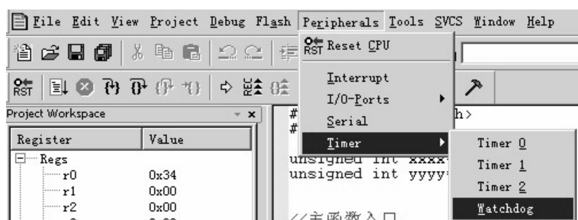


图 4-29 定时器菜单项

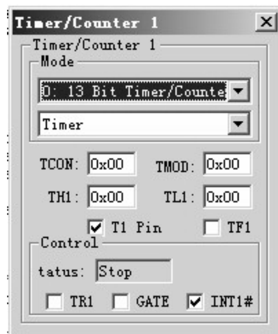


图 4-30 定时器设置窗口

（4）反汇编窗口

是反汇编窗口（Disassembly Windows）按钮，单击后可以把 C51 反汇编为相应的汇编语言，如果有汇编语言知识，就可以对比 C51 程序与汇编程序。由于汇编语言的效率高很多，这也可以作为查看 C51 执行效率的一种方法，详见案例 6。反汇编窗口如图 4-31 所示。

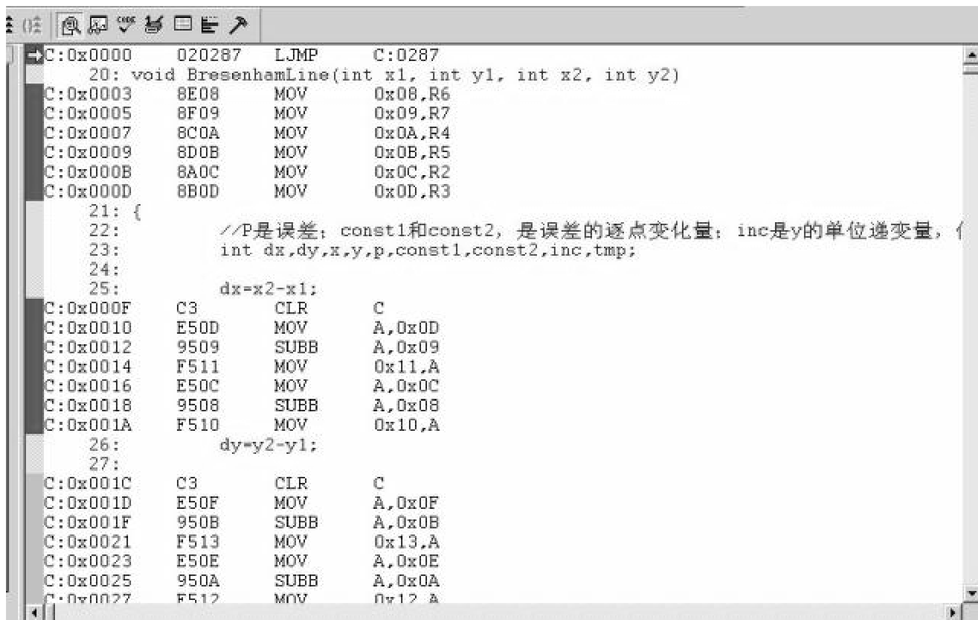


图 4-31　反汇编窗口

▷▷▷ 4.4.2　51 单片机中缩写的中英文速记

（1）51 指令

51 指令中英文速记见表 4-6～表 4-11。

表 4-6　数据传送类指令

指　令	英　文	中　文
MOV	Move	对内部数据寄存器 RAM 和特殊功能寄存器 SFR 的数据进行传送
MOVC	Move Code	读取程序存储器数据表格的数据传送
MOVX	Move External RAM	对外部 RAM 的数据传送
XCH	Exchange	字节交换
XCHD	Exchange Low-order Digit	低半字节交换
PUSH	Push onto Stack	入栈
POP	Pop from Stack	出栈

表 4-7　算术运算类指令

指　　令	英　　文	中　　文
ADD	Addition	加法
ADDC	Add with Carry	带进位加法
SUBB	Subtract with Borrow	带借位减法
DA	Decimal Adjust	十进制调整
INC	Increment	加 1
DEC	Decrement	减 1
MUL	Multiplication、Multiply	乘法
DIV	Division、Divide	除法

表 4-8　逻辑运算类指令

指令	英文	中文
ANL	And Logic	逻辑与
ORL	OR Logic	逻辑或
XRL	Exclusive-OR Logic	逻辑异或
CLR	Clear	清零
CPL	Complement	取反
RL	Rotate Left	循环左移
RLC	Rotate Left through The Carry Flag	带进位循环左移
RR	Rotate Right	循环右移
RRC	Rotate Right through The Carry Flag	带进位循环右移
SWAP	Swap	低 4 位与高 4 位交换

表 4-9　控制转移类指令

指　　令	英　　文	中　　文
ACALL	Absolute Subroutine Call	子程序绝对调用
LCALL	Long Subroutine Call	子程序长调用
RET	Return From Subroutine	子程序返回
RETI	Return from Interruption	中断返回
JMP	Jump Indirect	跳转
SJMP	Short Jump	短转移
AJMP	Absolute Jump	绝对转移
LJMP	Long Jump	长转移
CJNE	Compare and Jump if Not Equal	比较不相等则转移
DJNZ	Decrement and Jump if Not Zero	减 1 后不为 0 则转移
JZ	Jump if Zero	结果为 0 则转移
JNZ	Jump if Not Zero	结果不为 0 则转移
JC	Jump if the Carry Flag is Set	有进位则转移
JNC	Jump if Not Carry	无进位则转移

（续）

指　　令	英　　文	中　　文
JB	Jump if the Bit is Set	B 位为 1 则转移
JNB	Jump if the Bit is Not Set	B 位为 0 则转移
JBC	Jump if the Bit is Set and Clear the Bit	B 位为 1 则转移，并清除该位
NOP	No Operation	空操作

表 4-10　位操作指令

指　　令	英　　文	中　　文
SETB	Set Bit	置位

表 4-11　伪指令

指　　令	英　　文	中　　文
ORG	Origin	汇编起始地址
DB	Define Byte	定义数据字节
DW	Define Word	定义数据字
EQU	Equal	赋值
DATA	Data	数据（常表示地址信息）赋给字符
BIT	Bit	位定义
END	End	汇编结束

（2）51 外部引脚功能

51 外部引脚功能中英文速记见表 4-12。

表 4-12　51 外部引脚功能

名　　称	英　　文	中　　文
RST(9)	Reset	复位信号引脚
RxD(10)	Receive Data	串口接收端
TxD(11)	Transmit Data	串口发送端
INT0(12)	Interrupt0	外部中断 0 信号输入引脚
INT1(13)	Interrupt1	外部中断 1 信号输入引脚
T0(14)	Timer0	定时/计数器 0 输入信号引脚
T1(15)	Timer1	定时/计数器 1 输入信号引脚
WR(16)	Write	写信号引脚
RD(17)	Read	读信号引脚
PSEN(29)	Programmer Saving Enable	外部程序存储器读选通信号
ALE(30)	Address Latch Enable	地址锁存允许信号
EA(31)	Enable	外部 ROM 选择信号

（3）51 内部寄存器

51 内部寄存器中英文速记见表 4-13。

表 4-13　51 内部寄存器

名　称	英　文	中　文
SFR	Special Function Register	特殊功能寄存器
ACC	Accumulate	累加器
PSW	Programmer Status Word	程序状态字
CY(PSW.7)	Carry	进位标志位
AC (PSW.6)	Assistant Carry	辅助进位标志位
OV (PSW.2)	Overflow	溢出标志位
PC	Programmer Counter	程序计数器
DPTR	Data Point Register	数据指针寄存器
SP	Stack Point	堆栈指针
TCON	Timer Control	定时器控制寄存器
TF1(TCON.7)	Timer1 Flag	T1 中断标志位
TR1(TCON.6)	Timer1 Run	T1 运行控制位
TF0(TCON.5)	Timer0 Flag	T0 中断标志位
TR0(TCON.4)	Timer0 Run	T0 运行控制位
IE1(TCON.3)	Interrupt1 Exterior	外部中断 1 中断标志位
IT1(TCON.2)	Interrupt1 Touch	外部中断 1 触发方式选择位
IE0(TCON.1)	Interrupt0 Exterior	外部中断 0 中断标志位
IT0(TCON.0)	Interrupt0 Touch	外部中断 0 触发方式选择位
IE	Interrupt Enable	中断允许寄存器
EA(IE.7)	Enable All Interrupt	中断总允许位
ES(IE.4)	Enable Serial	串行口中断允许位
ET1(IE.3)	Enable Timer 1	T1 中断允许位
EX1(IE.2)	Enable Exterior 1	外部中断 1 中断允许位
ET0(IE.1)	Enable Timer 0	T0 中断允许位
EX0(IE.0)	Enable Exterior 0	外部中断 0 中断允许位
IP	Interrupt Priority	中断优先级寄存器
PS(IP.4)	Priority Serial	串口优先级标志位
PT1(IP.3)	Priority Timer 1	定时器 1 优先级标志位
PX1(IP.2)	Priority Exterior 1	外部中断 1 优先级标志位
PT0(IP.1)	Priority Timer 0	定时器 0 优先级标志位
PX0(IP.0)	Priority Exterior 0	外部中断 0 优先级标志位
PCON	Power Control	电源控制和波特率选择
TMOD	Timer Mode	定时器方式控制寄存器

（4）其他缩写

其他缩写中英文见表 4-14。

表 4-14　其他缩写

名　　称	英　　文	中　　文
MSB	Most Significant Bit	最高有效位
LSB	Last Significant Bit	最低有效位
OE	Output Enable	输出使能

4.4.3　51 存储器总结

存储器类型如图 4-32 所示，其总结见表 4-15。

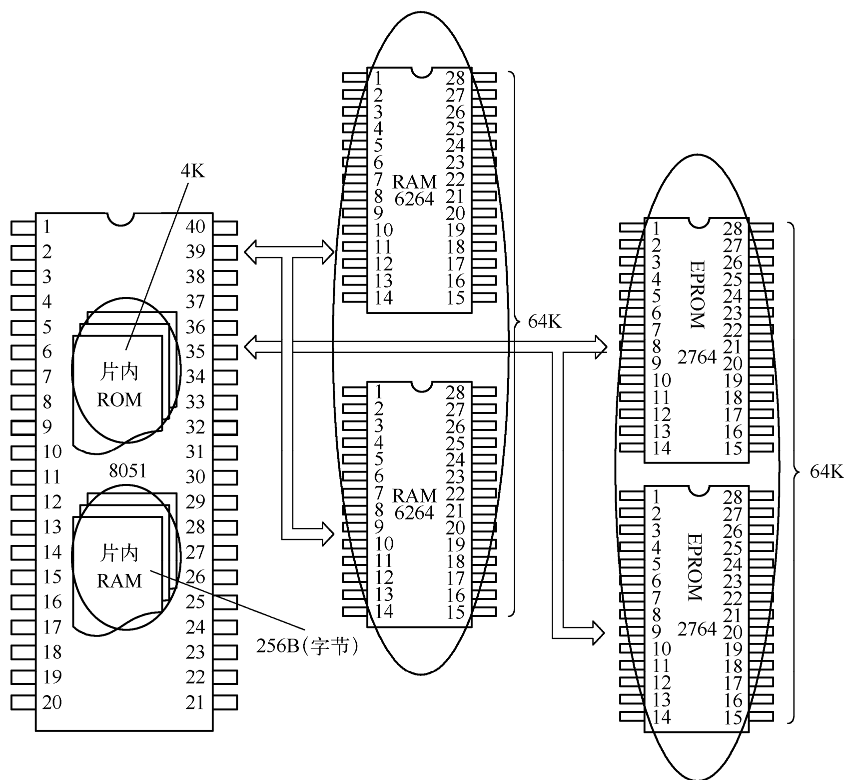


图 4-32　存储器类型

表 4-15　存储器总结

	容　　量	地　　址		寻 址 方 式	操 作 例 子
		字 节 地 址	位　地　址		
外部程序存储器	最大 64KB	0000H~FFFFH	无	PC 间址，DPTR 间址	MOVC A,@A+DPTR
外部数据存储器	最小 64KB	0000H~FFFFH	无	Ri 间址，DPTR 间址	MOVX A,@R0; MOVX @DPTR,A
内部数据存储器	128B	00H~7FH	00H~7FH	直接寻址，Ri 间址，位寻址	MOV 50H,#32H MOV @R1,A SETB 44H
特殊功能寄存器	21B	80H~FFH（不连续）	80H~FFH（不连续）	直接寻址，位寻址	MOV A,90H CLR B0H

▷▷▷ 4.4.4 寻址方式总结

由于寻址方式的复杂，初学者往往不易掌握数据在硬件中的真实位置，通过按物理地址为类别对寻址方式进行分类，将硬件（物理地址）和软件程序（寻址方式）结合起来，可以清晰地理解所寻址数据真实的位置。按物理地址对寻址方式进行分类见表 4-16。

表 4-16　按物理地址对寻址方式进行分类

物理地址单元	寻 址 方 式	举　　例
立即数	立即寻址	MOV A,#data
寄存器	寄存器寻址	INC R0
存储单元 RAM（寄存器除外）/ROM	直接寻址	MOV A,70H
	寄存器寻址	MOV A,@R0 MOVX A,@DPTR
	变址寻址	MOVC A,@ DPTR+A
	相对寻址	JC LOOP1
位	位寻址	MOV C,P1.0
I/O 接口	同外 RAM 寻址	MOVX A,@DPTR

▷▷▷ 4.4.5 PSW 总结

程序状态字寄存器 PSW 占 8 位。其各位含义见表 4-17。

表 4-17　PSW 各位含义

PSW7	PSW6	PSW5	PSW4	PSW3	PSW2	PSW1	PSW0
CY	AC	F0	RS1	RS0	OV		P
进位标志位	辅助进位标志位	用户标志位	寄存器选择位		溢出标志位		奇偶标志位

1）P：奇偶标志位。当 A 中 1 的个数为偶数时为 0；当 A 中 1 的个数为奇数时为 1。

2）OV：溢出标志位。当运算结果超带符号数范围时为 1；当运算结果不超带符号数范围时为 0。

3）RS0、RS1：寄存器选择位。见工作寄存器区。

4）AC：辅助进位标志位。当 D3 向 D4 无进位/借位时为 0；当 D3 向 D4 有进位/借位时为 1。

5）CY：进位标志位。当最高位无进位/借位时为 0；当最高位有进位/借位时为 1。

PSW 的位名称见表 4-18。

表 4-18　PSW 的位名称

位名称	CY	AC	F0	RS1	RS0	OV		P
单元地址加位	D0H.7	D0H.6	D0H.5	D0H.4	D0H.3	D0H.2	D0H.1	D0H.0
直接使用位地址	D7H	D6H	D5H	D4H	D3H	D2H	D1H	D0H
专用寄存器加位	PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0

算术运算指令对 PSW 标志位的影响见表 4-19。

表 4-19　算术运算指令对 PSW 标志位的影响

	ADD	ADDC	SUBB	DA	MUL	DIV
CY	√	√	√	√	0	0
AC	√	√	√	√	×	×
OV	√	√	√	×	√	√
P	√	√	√	√	√	√

表 4-19 中，符号“√”表示相应的指令操作影响标志；符号“0”表示相应的指令操作对该标志清 0；符号“×”表示相应的指令操作不影响标志。另外，增 1（INC A）和减 1（DEC A）指令影响 P 标志，但不影响其他标志位。

进位（借位）标志 CY 为无符号整数的多字节加法、减法、移位等操作提供了方便；溢出标志 OV 可方便地控制补码运算；辅助进位标志 AC 用于 BCD 码运算。算术运算操作将影响 PSW 中的 OV、CY、AC 和 P 等。

学过算术运算指令后，大家都对这些指令影响 PSW 寄存器有了印象。为什么要有 PSW 呢？原来这是由计算机的字长决定的。51 系列单片机是 8 位机，现在的 PC 是 64 位机。以后无论硬件如何发展，计算机的字长总是有限的，而人们对计算任务的精度的要求是无限的，有限的字长无法满足无限精度的要求，于是在计算中就出现了 PSW 寄存器，它是对硬件功能局限性的补充。有了 PSW 寄存器，51 单片机可以突破 8 位字长限制完成多字节加、减、乘、除运算，从而满足了任意精度的要求。这也是硬件和软件统一，硬件决定软件，软件依赖硬件起作用的一种体现。

▷▷▷ 4.4.6　有关 DPTR 的指令总结

在 8051 标准指令集的 111 条指令中，与 DPTR 有关的指令共有 5 类，分别为：

1）程序存储器查表指令，如“MOVC A,@A+DPTR”。

2）片外 RAM 传送指令，如“MOVX A,@DPTR”和“MOVX @DPTR,A”。

3）寄存器数据传送指令，即可对 DPTR 进行读写操作，在 8051 中 DPTR 由 DPH（DPTR 高 8 位字节）和 DPL（DPTR 低 8 位字节）构成，且 DPH 和 DPL 与一般的 SFR 一样，都可作为寄存器进行读写、压栈等操作。

4）程序转移指令，如“JMP @A+DPTR”。

5）运算指令，可分别对 DPH 和 DPL 进行运算操作。

通过对以上与 DPTR 相关的 5 类指令分析可知：第 3）类指令和第 5）类指令是将 DPTR 作为 SFR 进行操作的。第 1）类指令和第 4）类指令都是 DPTR 与 PC 指针进行的数据传送操作；第 2）类指令是对片外 RAM 地址寄存器进行的数据传送操作。因此，DPTR 的操作具体涉及 8051 中以下 3 个模块：SFR 读写模块、PC 指针模块及片外 RAM 地址模块。

经过这样的归纳，读者一定会对指令使用更加清晰，并且这种方法的关键是要亲自动手实践。

▷▷▷ 4.4.7　控制转移指令总结

控制转移指令基本分类如图 4-33 所示。

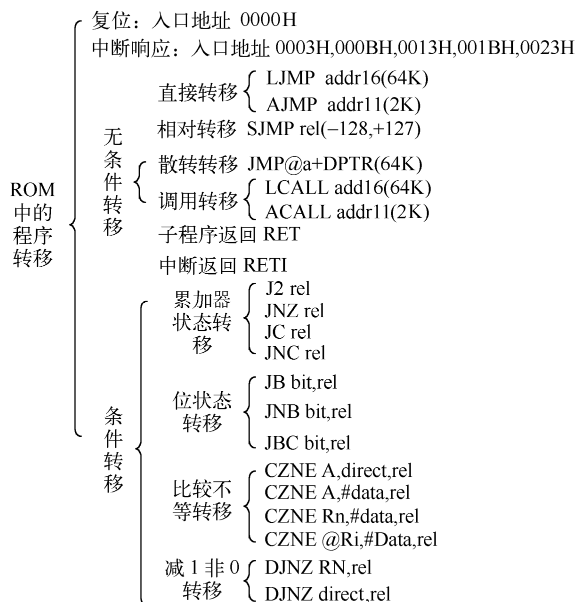


图 4-33　控制转移指令基本分类

案例 5

 程序的结构：分支和循环

▷▷ 5.1 案例任务

任务 1（分支程序）：在 2000H、2001H、2002H 首先存入 0、1、2 这 3 个数；然后判断 2000H～2002H 中有几个 0；把个数放在 2100H 单元中，并利用 Keil 的存储器窗口查看中间过程及结果。

任务 2（循环程序）：将 2000H～2002H 的内容清零，并利用 Keil 的存储器窗口查看中间过程及结果。

▷▷ 5.2 案例要点

（1）程序设计的步骤

1）分析任务，确定算法或解题思路。

2）按功能划分模块，确定各模块之间的相互关系及参数传递。

3）根据算法和解题思路画出程序流程图。

4）合理分配寄存器和存储器单元，编写汇编语言源程序（以“.ASM”扩展名保存），并进行必要的注释，以方便阅读、调试和修改。

5）将汇编语言源程序进行汇编和连接，生成可执行的目标文件（“.BIN 或.HEX”）。

6）仿真调试、修改，直至满足任务要求（仿真调试可以用软件模拟仿真，也可用硬件仿真）。

7）将调试好的目标文件（“.BIN 或.HEX”）烧录进单片机内，上电运行。

（2）分支程序的基本形式及设计要点

1）分支程序有两种基本形式，如图 5-1 所示。

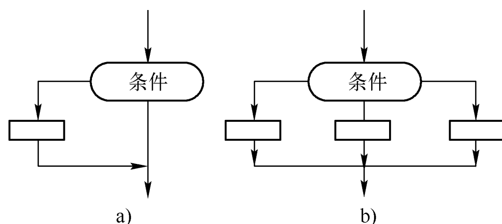


图 5-1　分支结构

2）分支程序的设计要点如下：

① 建立可供条件转移指令判断的条件。

② 选用合适的条件转移指令。

③ 在转移的目的地址处设定标号。

（3）循环程序

1）循环程序一般包括如下四个部分：初始化、循环体、循环控制、循环判断。

① 循环初始化：即确定循环开始时的状态，包括循环体变量初态和循环控制条件初态，通常为工作单元清零，寄存器和计数器置初值等。

② 循环体：重复执行的程序段。

③ 循环控制：循环条件修改，指修改计数器和修改内存指针。

④ 循环判断：根据循环次数或循环结束条件判断是否结束循环。

2）循环程序有先处理后判断和先判断后处理两种结构类型，如图 5-2 所示，两者功能一样，但在条件不满足的情况下，先处理后判断结构可执行一次。

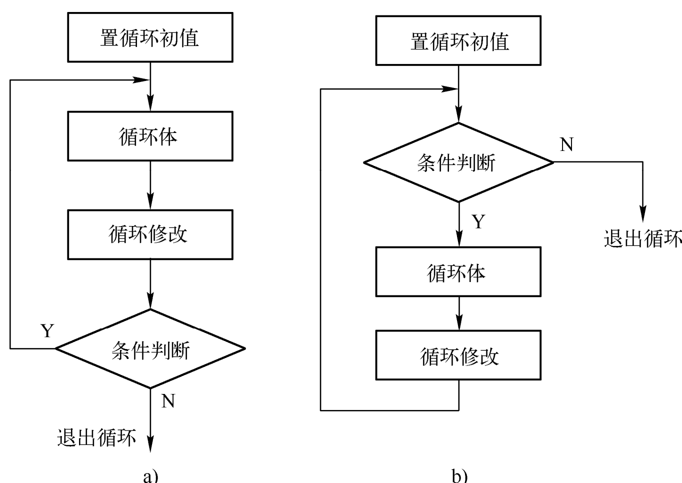


图 5-2　循环程序结构类型

a) 先处理后判断　b) 先判断后处理

3）循环程序按结构形式，有单重循环与多重循环。

① 在多重循环中，只允许外重循环嵌套内重循环。

② 不允许循环相互交叉，也不允许从循环程序的外部跳入循环程序的内部。

4）延时程序延时时间的计算。

5.3　案例设计

（1）编写任务 1 程序

1）汇编程序如下：

```
        ORG     0000H
MAIN:   MOV     DPTR,#2000H
        MOV     A,#00H
        MOV     R0,#3H
LOOP:   MOVX    @DPTR,A
```

```
        INC     DPTR
        INC     A
        DJNZ    R0,LOOP
L00:    MOV     R0,#03H
        MOV     R1,#00H
        MOV     DPTR,#2000H
L11:    MOVX    A,@DPTR
        CJNE    A,#00H,L16
        INC     R1
L16:    INC     DPTR
        DJNZ    R0,L11
        MOV     DPTR,#2100H
        MOV     A,R1
        MOVX    @DPTR,A
L1E:    SJMP    L1E
        END
```

程序中的 LOOP 循环执行后，可利用 x:2000H 查看外部 RAM 存储单元内容，如图 5-3 所示。由图中可见，0、1、2 已分别存入 2000H、2001H、2002H 单元。

程序全部执行完后，利用 x:2100H 查看外部 RAM 存储单元内容，如图 5-4 所示。由于 2000H～2002H 单元中只有 1 个单元保存 0，因此，外部 RAM 中 2100H 单元保存 1。

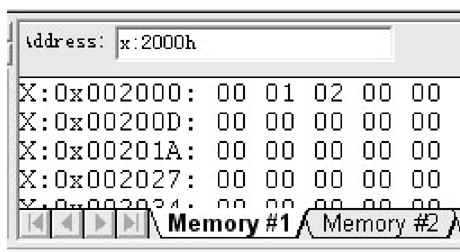


图 5-3　任务 1 存储器窗口显示 1

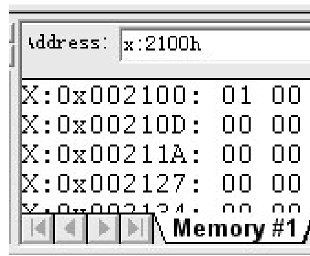


图 5-4　任务 1 存储器窗口显示 2

2）C 语言程序如下：

```
#include <reg51.h>
main ( )
{ unsigned char xdata *p=0x2000;          /*指针 p 指向 2000H 单元*/
  int n=0,i;
  for(i=0;i<3;i++)
  { if(*p= =0) n++;                       /* 若该单元内容为零，则 n+1 */
    p++;                                  /* 指针指向下一单元 */
  }
  p=0x2100;                               /* 指针 p 指向 2100H 单元 */
  *p=n;                                   /* 把个数放在 2100H 单元中 */
}
```

（2）编写任务 2 程序

1）汇编程序如下：

```
        ORG     0000H
MAIN:   MOV     DPTR,#2000H
        MOV     A,#00H
        MOV     R0,#3H
LOOP:   MOVX    @DPTR,A
        INC     DPTR
        INC     A
        DJNZ    R0,LOOP
SE01:   MOV     R0,#00H
        MOV     DPTR,#2000H
LOO1:   CLR     A
        MOVX    @DPTR,A
        INC     DPTR
        INC     R0
        CJNE    R0,#03H,LOO1
LOOP1:  SJMP    LOOP1
        END
L1E:    SJMP    L1E
        END
```

程序中的 LOOP 循环执行后，可利用 x:2000H 查看外部 RAM 存储单元内容，如图 5-5 所示。由图中可见，0、1、2 已分别存入 2000H、2001H、2002H 单元。

程序全部执行完后，利用 x:2000H 查看外部 RAM 存储单元内容，如图 5-6 所示。说明 2000H～2002H 单元已全部清零。

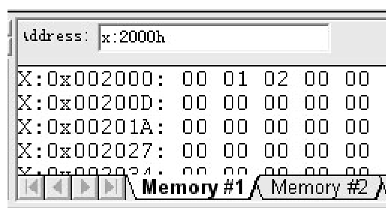


图 5-5　任务 2 存储器窗口显示 1

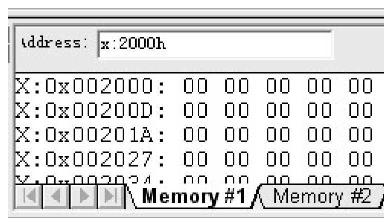


图 5-6　任务 2 存储器窗口显示 2

2）C 语言程序如下：

```
#include <reg51.h>
main( )
  {
     int     i;
     unsigned char xdata *p=0x2000;
                                          /* 指针指向 2000H 单元 */
     for(i=0;i<3;i++)
     {*p=0; p++;}                         /*清零 2000H～2002H 单元*/
  }
```

5.4 案例笔记

5.4.1 程序设计的方法

（1）评价程序优劣的因素

1）正确性，容错性。

2）结构化，简明易读，易检验，易维护。

3）省资源，高效率，易操作。

（2）程序设计过程与基本设计方法

1）模块化；结构化；自顶而下与自底而上。

2）结构化设计。

3）结构设计自顶而下：

功能设计→总体结构设计→局部结构设计→底层模块设计→验证方法设计

具体设计自底而上：

模块→局部→整体，逐步整合、协调，调试与验证，最后总结建档。

4）从原理到程序实现：

原理→模型→算法→流程→程序设计→调试→优化→验证→建档。

（3）常用的程序调试方法

1）原则：先硬后软；先局部，后整体。

2）汇编检错，语法检查。

3）审视推演，逻辑检查。

4）准备测试数据，试运行。

5）附加测试指令，设置标志，输出中间结果。

6）单步调试。

7）设置断点。

（4）程序的优化与文件编制

1）参照评价因素，修改程序结构、数据结构、算法及程序等。

2）总结建档，编制说明文件。

3）设计说明：

设计目标，原理，模型；设计方案，性能与特点；程序结构，数据结构，存储器分配；流程，加注释的程序清单等。

4）测试报告：

测试方法、测试数据、测试结果分析。

5）使用说明

功能、操作方法、出错信息与排除方法、注意事项等。

5.4.2 汇编语言常见错误总结

1）标号重复。常见于复制、粘贴程序时忘记修改标号，造成出现多个相同的标号，标

号是不允许重复的。

2）标点符号以全角方式输入。汇编程序要求标点符号为半角方式，否则汇编失败。可以在输入:,;时切换到半角方式，或者在大写状态输入标点符号。这也是很容易犯而且不容易发觉的错误。

3）数值＃FFH 前遗漏0。根据要求应该在 a～f 前加 0，写成＃0FFH。

4）字母 O 和数字 0 混淆。

5）标号后边遗漏"："。

6）标号使用了特殊字符。标号不能用指令助记符、伪指令、特殊功能寄存器名和 8051 在指令系统中用的"#"、"@"等，长度以 2~6 字符为宜，第一字母必须是英文字母。例如，T1、T2、A、B 这些字符有特定的含义，不允许用于标号。

7）AJMP 跳转超过 2KB 地址。AJMP 属于短跳转命令，有 2KB 地址范围的限制。

8）超过地址范围。JB P3.2,EXIT 跳转超过-128～127 个地址范围。例如：

```
JB      P3.2,EXIT
```

建议修改为

```
JNB     P3.2,LD01
AJMP    EXIT
LD01: AJMP      EXIT
……
```

9）字母 I 和数字 1 混淆。

10）创造发明不存在的汇编语言指令。这种指令汇编程序不支持，芯片也不认可。

11）寄存器重复调用。例如，主程序中设定了 R4=5，表示主程序循环执行 5 次，而其中的一个延时子程序又用到 R4，使 R4 的值发生紊乱，造成程序无法正常执行。

12）硬件不熟悉。单片机一般采用下拉输出，往往端口输出 0 驱动外设工作，和常见的正逻辑相反，容易搞错。

编写汇编语言的忠告：

要养成良好的程序书写习惯，如标号对齐、参数对齐、注释对齐，这样看起来赏心悦目，也不容易出错。标号最好采用有意义的英文，这样比较直观，注释尽量详细准确，便于以后读懂，而且有利于其他程序中作为子程序模块的调用。还有要注意典型程序模块的积累，再复杂的程序也是由一个个小程序模块组成的，在初学阶段可以对典型程序如延时子程序、查表子程序、按键消抖子程序等编写实践一次，这样印象深刻，便于以后引用。

▷▷▷ 5.4.3 延时程序延时时间计算

【例 5-1】 51 单片机晶振为 12MHz，编写 10ms 延时子程序。12MHz 的机器周期为 1μs。

利用循环结构实现的延时子程序如下：

```
        ORG     1000H
DEL:    MOV     R7,#40          ;单周期 1μs，外循环初值
DEL1:   MOV     R6,#125         ;单周期 1μs，循环初值
DEL2:   DJNZ    R6,DEL2         ;双周期 2μs
```

```
        DJNZ    R7,DEL1         ;双周期 2μs
        RET;                    ;双周期 2μs
```

定时时间公式通常为

定时时间=T 初值+(T 循环体+T 循环修改+T 循环判断)×循环次数

根据定时时间公式，可得

内环定时时间：1+2×125μs =251μs

外环定时时间：1+(251+2)×40+2μs =10123μs=10.123ms

若要实现精确定时，可将程序修改如下：

```
        ORG     1000H
DEL:    MOV     R7,#40          ;单周期 1μs，外循环初值
DEL1:   MOV     R6,#123         ;单周期 1μs，内循环初值
        NOP                     ;单周期 1μs
DEL2:   DJNZ    R6,DEL2         ;双周期 2μs
        DJNZ    R7,DEL1         ;双周期 2μs
        RET;                    ;双周期 2μs
```

内环定时时间：2+2×123μs =248μs

外环定时时间：1+(248+2)×40+2μs =10001μs =10.003ms

案例 6



程序的效率



▷▷ 6.1 案例任务

通过对一个流水灯 C 语言程序的修改，利用 Keil 的存储器窗口查看中间过程及结果，体会影响程序效率的因素。

▷▷ 6.2 案例要点

1）按照编程的结构及其功能计算机程序设计语言可以分为三种：

① 机器语言。机器语言是用二进制代码 0 和 1 表示指令和数据的最原始的程序设计语言。

② 汇编语言。在汇编语言中，指令用助记符表示，地址、操作数可用标号、符号地址及字符等形式来描述。

③ 高级语言。高级语言是接近于人的自然语言，面向过程而独立于机器的通用语言。

2）熟悉汇编语言程序的基本结构类型、语法规则和常用的伪指令等。

3）能够根据算法绘制流程图，使用规范的流程图符号，常用元素见表 6-1。

表 6-1 流程图的常用元素

名　称	图　形	作　用	填 充 内 容
椭圆框	（椭圆框图形）	用于表示程序的开始或结束	使用时在框内标注中、英文“开始”或“结束”
矩形框	（矩形框图形）	用于说明一段程序功能	在框内用字符注明某段程序或某条指令的作用
菱形框	（菱形框图形）	用来进行判断以决定程序的走向	框内注明判断条件
圆	（圆形图形）	当流程图在一页上画不完时，为了确保流程图完整确，在相应的连接处画上相同的符号，以表示流程图从这里流向页外某个地方	在应予连接的两处程序框图处的圆框中标注相同的数字
带箭头的线段	→	表示程序的流向	在流程图中用它连接上述各种框图，以表明程序执行的顺序或可能的分支

4）掌握提高程序效率的方法。

▷▷ 6.3 案例设计

在 Keil 软件中输入如下 C 代码：

```
#include <reg52.h>                  //包含头文件
sbit LED=P2^0;                      //定义位变量发光二极管，使其关联单片机引脚 P2.0
void Delayms(unsigned int t);       //定义延时函数
int main(void)                      //主函数（C 语言程序入口函数）
{
        while(1)
        {
                LED=0;              //P2.0 拉低，点亮发光二极管
                Delayms(500);       //调用延时函数，延时 500ms
                LED=1;              //P2.0 拉高，熄灭发光二极管
                Delayms(500);       //调用延时函数，延时 500ms
        }
        return 0;
}
void Delayms(unsigned int t)        //延时函数
{
         unsigned int i,j;
         for(i=0;i<t;i++)
                 for(j=0;j<120;j++);     //大约延时 1ms
}
```

这是控制 P2.0 发光二极管闪烁的 C 源码，这个源码在 Keil uVision4 生成的程序代码是 67 个字节，如图 6-1 所示，图中 code=67 说明上述 C 代码编译后为 67 个字节。而这 67 个字节是编译后按照汇编语言的字节数进行计算的，如图 6-2 所示。下面就采用几种方法来提高这个程序的效率。

```
Build target 'Target 1'
compiling EX6.c...
linking...
Program Size: data=9.0 xdata=0 code=67
"EX6" - 0 Error(s), 0 Warning(s).
Build / Command / Find in Files
```

图 6-1　编译结果 1

```
     4: int main(void)//主函数（C语言程序入口函数）
     5: {
     6:         while(1)
     7:         {
     8:                 LED=0;//P2.0拉低，点亮发光二极管
C:0x0023    C2A0     CLR      LED(0xA0.0)
     9:                 Delayms(500);//调用延时函数，延时500毫秒
C:0x0025    7FF4     MOV      R7,#0xF4
C:0x0027    7E01     MOV      R6,#0x01
C:0x0029    120003   LCALL    Delayms(C:0003)
    10:                 LED=1;//P2.0拉高，熄灭发光二极管
C:0x002C    D2A0     SETB     LED(0xA0.0)
    11:                 Delayms(500);//调用延时函数，延时500毫秒
C:0x002E    7FF4     MOV      R7,#0xF4
C:0x0030    7E01     MOV      R6,#0x01
C:0x0032    120003   LCALL    Delayms(C:0003)
    12:         }
C:0x0035    80EC     SJMP     main(C:0023)
C:0x0037    787F     MOV      R0,#0x7F
C:0x0039    E4       CLR      A
C:0x003A    F6       MOV      @R0,A
C:0x003B    D8FD     DJNZ     R0,C:003A
C:0x003D    758107   MOV      SP(0x81),#0x07
C:0x0040    020023   LJMP     main(C:0023)
C:0x0043    00       NOP
EX6.c  Disassembly
```

图 6-2　反汇编结果

（1）尽量定义局部变量

单片机程序的全局变量一般是放在通用数据存储器（RAM）中，而局部变量一般是放在特殊功能寄存器中。上述 C 程序的执行状态如图 6-3 所示。图中的箭头指向下一条将要执行的指令。在 Delayms(500)中 i、j 都为局部变量，因此放在寄存器中；Delayms(500)执行后 j=120，由图 6-3 左边部分可知 r3=0x78=120，因此 j 保存在 r3 中；r4r5=0x01f4=500，因此 i 保存在 r4r5 中。

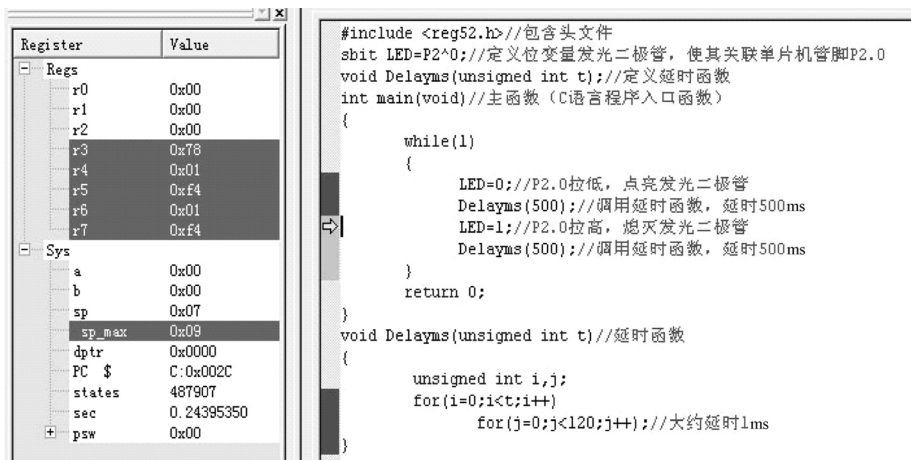


图 6-3　程序执行状态

处理寄存器数据的速度比处理 RAM 数据要快，如果在一个局部函数里调用一个全局变量将会多生成代码。所以，少定义全局变量，多定义局部变量。如上述 C 代码中，如果把延时函数里的 i 和 j 定义为全局变量，编译后程序代码会增加到 79 个字节，多了 12 个字节，如图 6-4 所示。

（2）省略函数定义

在一个单片机程序里我们习惯在 main 函数的前面先定义被调用函数，然后在 main 函数的下面再实现被调用函数。这样的写法固然是一个好习惯，但每定义一个函数会增加代码，而且函数形参数据类型越大，形参越多，增加的代码就越多。如果不定义，编译器又报错。由于 C 编译器的编译顺序是从上往下编译，只要被调用的函数在主调函数调用之前实现就没有问题了。所以，优化的写法是不用定义函数，但要按先后顺序（被调用函数一定要在主调函数之前写好）来写函数实现，最后再写 main 函数。这样做编译器不但不会报错，而且代码得到精简。如上述 C 代码中，把延时函数的定义删除了，然后把延时函数的实现搬到 main 函数的上面，编译后程序代码减少到 63 个字节，减少了 4 个字节，如图 6-5 所示。

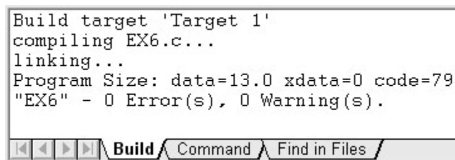


图 6-4　编译结果 2

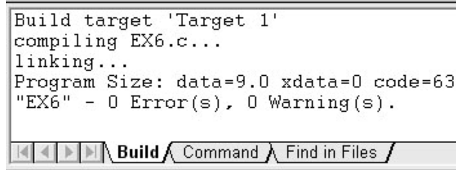


图 6-5　编译结果 3

（3）省略函数形参

函数带形参，是为了在函数调用时传递实参，不但可以避免重复代码出现，还可以通过传递不同的实参值多次调用函数且实现不同的函数功能，总体代码也会得到精简。在实际编程时，我们只要稍加注意，还可以进一步精简代码。对于不是多次调用或者多次调用但实参值不变的函数，可以省略函数形参。如上述 C 代码中的延时函数，把它改写成不带形参的函数：

```
void Delayms()                    //延时函数
{
    unsigned int i，j;
    for(i=0;i<500;i++)
        for(j=0;j<120;j++);       //大约延时 1ms
}
```

编译后，程序代码变成 56 个字节，精简了 11 个字节，如图 6-6 所示。

（4）改换运算符

在第（2）步的基础上进行修改。

C 运算符的运用也会影响程序代码的数量。如上述 C 代码中，把延时函数里的自加运算符改成自减运算符后，即

```
void Delayms(unsigned int t)          //延时函数
{
    unsigned int i，j;
    for(i=t;i>0;i− −)
        for(j=120;j>0;j− −);          //大约延时 1ms
}
```

编译后，程序代码变成 65 个字节，精简了 2 个字节，如图 6-7 所示。

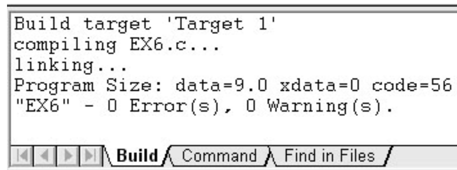

Build target 'Target 1'
compiling EX6.c...
linking...
Program Size: data=9.0 xdata=0 code=56
"EX6" - 0 Error(s), 0 Warning(s).

图 6-6　编译结果 4

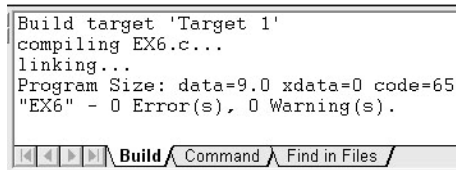

Build target 'Target 1'
compiling EX6.c...
linking...
Program Size: data=9.0 xdata=0 code=65
"EX6" - 0 Error(s), 0 Warning(s).

图 6-7　编译结果 5

（5）选择合适的数据类型

在第（2）步的基础上进行修改。

C 语言里选择变量的数据类型很讲究，变量的数据类型过小满足不了程序的要求，过大则会占用太多的 RAM 资源。数据类型定义也影响程序代码的大小，而且这个影响还比较大。如上述 C 代码中，延时函数里的局部变量 j 定义的数据类型明显偏大，如果把它由 unsigned int 改成 unsigned char，编译后，程序代码变成 55 个字节，精简了 12 个字节，如图 6-8 所示。

（6）直接嵌入代码

在程序里如果某个函数只调用一次，而又要求代码提高执行速度，建议不要采用调用函

数的形式，而应该将该函数里的代码直接嵌入主调函数里，代码执行效率会大大提高。

（7）使用效率高的 C 语句

C 语言里有一个三目运算符“？”，俗称“问号表达式”。很多程序员都很喜欢使用，因为它逻辑清晰、表达简洁。例如，c=(a>b) ? a+1 : b+1;实际上等效于以下的 if…else 结构：

```
Build target 'Target 1'
compiling EX6.c...
linking...
Program Size: data=9.0 xdata=0 code=55
"EX6" - 0 Error(s), 0 Warning(s).
Build  Command  Find in Files
```

图 6-8 编译结果 6

```
    if (a>b)
c=a+1;
    else
c=b+1;
```

可以看到，使用问号表达式，语句相当简洁，但执行效率很低，远没有 if…else 语句效率高。所以，当程序要求提高执行速度时，建议不要使用问号表达式。

另外，do-while 语句也比 while 语句的效率高。

代码的效率问题，不是编程中的主要问题，除了程序要求较高的执行速度或者单片机的 ROM 和 RAM 不够用时才会考虑。一般情况下，还要兼顾其他问题。如果仅追求高效率的代码，可能会影响代码的可读性和可维护性。

6.4 案例笔记

6.4.1 汇编语言与 C 语言

程序设计语言的层次结构如下：

硬件逻辑被虚拟化成汇编语句，汇编语句再次被封装，虚拟化成高级语言语句。高级语言的语句，再次被封装，形成一个特定目的的程序，或者称为函数，然后这些函数再通过互相调用，生成更复杂的函数，再将这些函数组合起来，就形成了最终的应用程序。程序再被操作系统虚拟成一个可执行文件。其实这个文件到了底层，就是一次一次地对 CPU 的电路信号刺激。也就是说，硬件电路逻辑，一层层地被虚拟化，最终虚拟成一个程序，程序就是对底层电路作用的一种表达形式。按照与硬件虚拟化关系的远近，计算机程序设计语言分为机器语言、汇编语言和高级语言，它们的关系如图 6-9 所示。

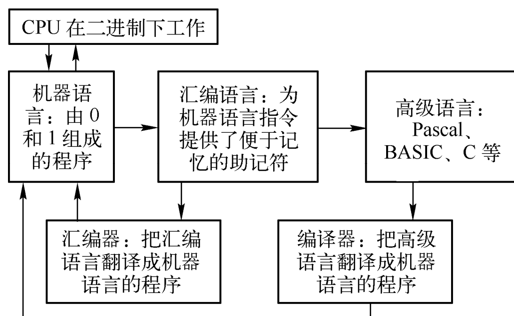


图 6-9 程序设计语言

汇编语言有如下缺点：

1）汇编语言与处理器密切相关。每种处理器都有自己的指令系统，相应的汇编语言各不相同。所以，汇编语言程序的通用性、可移植性较差。

2）汇编语言编写涉及寄存器、主存单元等硬件细节，所以编写程序比较繁琐，调试起

来也比较困难。

汇编语言有如下优点：

1）汇编语言是一种用文字助记符来表示机器指令的符号语言，是最接近机器码的一种语言，它可以直接、有效地控制计算机硬件。其主要优点是占用资源少，程序执行效率高，由于它一条指令就对应一条机器码，每一步的执行动作都很清楚。

2）懂得汇编语言可帮助了解任何影响语言效率的规则，并且程序大小和堆栈调用情况都容易控制。例如，懂得汇编语言指令就可以使用在片内 RAM 作变量的优势，因为片外变量需要几条指令才能设置累加器和进行数据指针存取。同样的，当要求使用浮点数和启用函数时也只有具备汇编编程经验才能避免生成庞大的、效率低的程序，对于这方面的编程，没有汇编语言是做不到的。

与汇编语言相比，C 语言有如下优点:

1）对单片机的指令系统不要求了解，仅要求对 51 单片机的存储器结构有初步了解，至于寄存器分配、不同存储器的寻址及数据类型等细节均由编译器管理。

2）程序有规范的结构，可分为不同的函数。这种方式可使程序结构化，将可变的选择与特殊操作组合在一起，改善了程序的可读性。

3）编程及程序调试时间显著缩短，从而提高了效率。提供的库包含许多标准子程序，具有较强的数据处理能力，在已编好程序中可容易地植入新程序，因为它具有方便的模块化编程技术。

4）单片机 C 语言作为一种非常方便的语言而得到广泛的支持，C 语言程序本身并不依赖于机器硬件系统，基本上不作修改就可根据单片机的不同较快地移植过来。

C 语言更新维护方便、可移植性强，但实践证明，单独使用 C 语言开发单片机程序也存在诸多不足，如占用存储空间大、对硬件控制不灵活等。因此想成为一个优秀的单片机编程技术员，最好就是能懂得单片机 C 语言和汇编语言的混合编程。在编程过程中，通常用 C 语言来构建程序框架，而用汇编程序作为子程序来处理一些有实时性要求的特殊应用。与硬件关系密切的程序或对性能有特殊要求的程序往往用汇编语言设计，上层应用软件则往往用 C 语言来设计。

作为单片机初学者，还是应该先学习汇编语言，因为汇编语言程序除了具有简洁明快、跳跃性强、占 ROM 资源少等优点以外，还因它和单片机底层硬件紧密联系，可以让初学者更加了解单片机硬件系统各种资源、软件的工作原理，熟悉各个功能模块的作用，从而为编出更高效率的程序打好扎实的基础。

▷▷▷ 6.4.2 C 语言常见错误总结

1）使用非法的或错误的标识符，如 main 写成 mian，printf()写成 print()，或使用 sin(2α)、cosΦ、π*r*r 等。

2）变量未经定义就使用。

3）变量类型使用不当（取值范围不够大、本该用整型而用了实型、精度不够等）。

4）变量未经初始化就在表达式中使用。

5）语句或定义结束缺少分号或误用分号（复合语句结束后面不需要分号）。

6）表达式中漏写了必要的乘号*。例如，将 3*x*y＋5 错写成 3xy＋5。

7）表达式中缺少必要的圆括号，或圆括号不匹配，或者用花括号、方括号取代了圆括号。

8）忘记了注释的结束符*/；正确的应是以/*开始，以*/结束，所用的两个符号“*”和“/”之间不能用空格隔开。



9）在该用小写字母的地方，却用了大写字母（例如，把 main 写成 Main、scanf 写成 Scanf；定义变量名是小写，但在程序中却用了大写的变量名，其中 s、c、x、k、z 最易用错，如 s1 写成 S1、ch 写成 Ch）。

10）在语句之间对变量进行了定义。正确方法是在函数体中将所有定义放在所有语句之前。

11）编写代码（程序）时就特别要注意，避免程序在运行时，用 0 作为除数。

12）在字符串或输入/输出格式控制串外的其他地方，用了非法的标点符号（除了英文半角输入法，其他输入方式下的标点符号都是不对的）。

13）漏写函数体结束时的花括号，或者花括号不配对。

14）分隔符使用不正确，例如 inta,b　c .d; 应为 int a,b,c,d;。

15）程序中调用了库函数，但忘记包含相应的头文件（如要包含头文件：math.h）。

16）标准输入/输出头文件包含时出错，正确的是#include<stdio.h>或者#include “stdio.h”，但有很多读者会出现拼写错误。

案例7

 流　水　灯

7.1　案例任务

P1.4～P1.7 所接发光二极管顺次点亮，首先将 P1.4 口的 L3 点亮，其他发光二极管熄灭，然后 L3 灭 L4 亮，L4 灭 L5 亮，L5 灭 L6 亮，L6 灭 L3 亮，循环执行。要求使用位操作指令法、左右移指令法、查表法三种方法实现。

7.2　案例要点

（1）单片机 4 组 I/O 口 P0～P3 内部结构、对应的特殊功能寄存器、驱动能力及程序控制方法

（2）每个端口都包括锁存器、输出驱动器、两个三态缓冲器以及控制电路

（3）P0 口特点

1）控制端高电平时，作为低 8 位地址和 8 位数据分时使用口，供扩展时使用。

2）控制端低电平时，接地端 MOS 管截止，使非接地端 MOS 管漏极开路，输出“1”时须外接上拉电阻，最小系统作准双向 I/O 口用。

3）对应特殊功能寄存器地址 80H。

（4）P1 口特点

1）准双向口：作为 I/O 输入时，口锁存器必须置“1”，使 MOS 管截止，输入信号通过“读引脚”三态缓冲器进入内部总线。

2）内部有上拉电阻（20～40kΩ）。

3）由于没有其他复用，在应用时，是 I/O 优先选用端口。

4）对应特殊功能寄存器地址 90H。

（5）P2 口特点

1）控制端高电平时，作为高 8 位地址输出口。

2）控制端低电平时，最小系统（8051、8751）作准双向 I/O 口用。

3）对应特殊功能寄存器地址 A0H。

（6）P3 口特点

1）准双向口：条件为第二功能输出端常“1”，与门开锁。

2）第二功能口：作为第二功能口使用时，（P3）=FFH；某位作为第二功能输入时，第二功能输出也必须置“1”。

P3 口引脚的第二功能见表 7-1。

表 7-1 P3 口第 2 功能

引 脚	第 二 功 能	引 脚	第 二 功 能
P3.0	RXD（串行数据输入）	P3.4	T0（定时器 0 外部输入）
P3.1	TXD（串行数据输出）	P3.5	T1（定时器 1 外部输入）
P3.2	INT0（外部中断 0 输入）	P3.6	WR（外部 RAM 写信号）
P3.3	INT1（外部中断 1 输入）	P3.7	RD（外部 RAM 读信号）

3）对应特殊功能寄存器地址 B0H。

（7）读锁存器与读引脚

80C51 单片机中，输入有两种方式，分别称为“读引脚”和“读锁存器”。

第一种方式是将引脚作为输入，从外部引脚读进输入的值，即当引脚作为输入端时用读引脚的方式来输入。

第二种方式是引脚作为输出端使用时采用的工作方式。

8051 单片机中引入了读锁存器这种操作，避免出现“失误”，读的是控制锁存器，而不是引脚本身。

（8）掌握位操作指令、移位指令、查表程序设计

（9）发光二极管结构、发光原理及程序驱动

▷▷ 7.3 案例设计

▷▷▷ 7.3.1 硬件电路

流水灯硬件电路如图 7-1 所示。

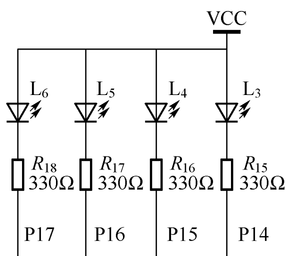


图 7-1 流水灯硬件电路

▷▷▷ 7.3.2 软件代码

1．查表法

（1）汇编程序

```
ORG     0000H          ;单片机上电后从 0000H 地址执行
JMP     START          ;跳转到主程序存放地址处
```

```
            ORG      0030H              ;设置主程序开始地址
START:      MOV      SP,#60H            ;设置堆栈起始地址为 60H
            MOV      DPTR,#TAB          ;表格首地址送 DPTR
LOOP:       CLR      A                  ;累加器清零
            MOVC     A, @A+DPTR         ;取数据表中的值
            CJNE     A ,#0FFH,SHOW      ;检查发光二极管顺次点亮结束标志
            AJMP     START              ;从头开始点亮
SHOW:       MOV      P1,A               ;将数据送到 P1 口
            ACALL    DELAY              ;调用延时子程序
            INC      DPTR               ;取数据表指针指向下一数据
            AJMP     LOOP               ;继续查表取数据
DELAY:      MOV      R0,#0FFH           ;延时子程序
D1:         MOV      R1,#0FFH
            DJNZ     R1,$
            DJNZ     R0,D1
            RET
TAB:        DB 0EFH,0DFH,0BFH,07FH,0FFH
            END
```

（2）C 语言程序

```
//查表法
#include<AT89X51.h>
void main (void)
{
unsigned    char LED[4]={0xef,0xdf,0xbf,0x7f};
unsigned char i;
unsigned int a;
do{
for (i=0;i<4;i++)
{
P1=LED[i];
for (a=0;a<30000;a++);
}
}
while (1);
}
```

2．位操作

（1）汇编程序

```
            ORG      0000H              ;单片机上电后从 0000H 地址执行
            JMP      START              ;跳转到主程序存放地址处
            ORG      0030H              ;设置主程序开始地址
START:      MOV      SP,#60H            ;设置堆栈起始地址为 60H
            CLR      P1.4               ;P1.4 输出低电平，使 LED3 点亮
            ACALL    DELAY              ;调用延时子程序
```



```
        SETB    P1.4        ;P1.4 输出高电平，LED3 熄灭
        CLR     P1.5        ;P1.5 输出低电平，LED4 点亮
        ACALL   DELAY       ;调用延时子程序
        SETB    P1.5
        CLR     P1.6
        ACALL   DELAY
        SETB    P1.6
        CLR     P1.7
        ACALL   DELAY
        SETB    P1.7
        AJMP    START
DELAY:  MOV     R0,#0ffh    ;延时子程序
D1:     MOV     R1,#0ffh
        DJNZ    R1 ,$
        DJNZ    R0 ,D1
        RET
        END
```

（2）C 语言程序

```
//位操作法
#include<AT89X51.h>
void main (void)
{
unsigned int i;            //定义变量 i
P1=0xff;                   //全灭
do{
for( i=0;i<30000;i++)
P1_4=0;
P1_4=1;
for(i=0;i<30000;i++)
P1_5=0;
P1_5=1;
for(i=0;i<30000;i++)
P1_6=0;
P1_6=1;
for(i=0;i<30000;i++)
P1_7=0;
P1_7=1;
}while (1) ;
}
```

3．移位指令

（1）汇编程序

```
        ORG     0000H       ;单片机上电后从 0000H 地址执行
        JMP     START       ;跳转到主程序存放地址处
        ORG     0030H       ;设置主程序开始地址
```

```
START:    MOV     SP,#60H          ;设置堆栈起始地址为 60H
LOOP:     MOV     A,#0EFH          ;ACC 中先装入 LED3 亮的数据
LOOP1:    MOV     P1,A
          RL      A                ;将 ACC 中的数据左移
          CALL    DELAY
          CJNE    A,#07FH,LOOP1
          ANL     P1,A
          JMP     LOOP
DELAY:    MOV     R0,#250
D1:       MOV     R1,#200
          DJNZ    R1,$
          DJNZ    R0,D1
          RET
          END
```

（2）C 语言程序

```
//移位指令程序
#include<AT89X51.h>
void main (void)
{
unsigned int i;
unsigned char a;
unsigned char b;
do{
b=0xef;
for(a=0;a<4;a++)
{
     P1=b;
  for(i=0;i<30000;i++) ;
  b<<=1;
   }
 }
 while (1);
 }
```

7.4 案例笔记

7.4.1 51 流水灯电路在 C51 学习中的应用

通过本案例中的硬件电路，结合如下程序可学习 C 语言的常用运算符。

（1）sbit 定义

```
/*
 * 位定义实现 3 个 LED 的点亮
 */
```

```
#include <reg52.h>

#define    ON 0
#define    OFF 1

sbit led1=P1^0;
sbit led2=P1^2;
sbit led3=P1^3;
main()
{
        bit a;
        a=ON;

        led1=led2=led3=a;
        while(1);
}
```

（2）char 定义

```
/*
 * char 变量的用法
 */
#include <reg52.h>

#define unchar      unsigned char

main()
{
        unchar led;
        led=0x05;

        P1 = led;
        while(1);
}
```

（3）*的使用

```
/*
 * *的使用
 */
#include <reg52.h>

#define unchar      unsigned char

main()
{
        unchar led;
        led = 0x00 + 1*8;
```



```
        P1 = led;
        while(1);
}
```

（4）<<运算符

```
 /*
 *  复合赋值运算符的使用
 */
#include <reg52.h>

#define unchar    unsigned char
#define led3        (1<<3)

main()
{
/*      unchar led = 0;
        led |= led6 ;         //置位
*/
        unchar led = 0xff;
        led &= ~led3;

        P1 = led;
        while(1);
}
```

（5）+的使用

```
/*
 * +的使用
 */
#include <reg52.h>

#define unchar      unsigned char

main()
{
        unchar led;
        led=0x00+0x08;

        P1 = led;
        while(1);
}
```

（6）%的使用

```
/*
 * %的使用
```

```
 */
#include <reg52.h>

#define unchar    unsigned char

main()
{
        unchar led;
        led = 66%5;

        P1 = led;
        while(1);
}
```



（7）<<的使用

```
/*
 * <<的使用
 */
#include <reg52.h>

#define unchar    unsigned char

main()
{
        unchar led;
        led = 0x10 >>3;

        P1 = led;
        while(1);
}
```

（8）&的使用

```
/*
 * &的使用
 */
#include <reg52.h>

#define unchar    unsigned char

main()
{
        unchar led;
        led = 0x05 & 0x0A;

        P1 = led;
        while(1);
}
```

▷▷▷ 7.4.2　C51 双向口和准双向口

51 单片机的说明书上："Because Ports 1, 2, and 3 have fixed internal pull ups, they are sometimes called quasi-bidirectional ports. When configured as inputs, they pull high and source current (IIL) when externally pulled low. Port 0, on the other hand, is considered truly bidirectional, because it floats when configured as an input. "

翻译是：因为 P1、P2、P3 有固定的内部上拉电阻，所以有时称它们为准双向口。当用做输入时被拉高，若要求低电平则要靠外部电路拉低。而 P0 则是真双向口，因为作为输入时它是悬浮的。“准”就是“基本上的意思”，也就是“准双向口”不是真正的双向口。

其实重点在 P0 口。P0 口的双向指的是它被用做地址/数据端口时，才处于两个开关管推挽状态，当两个开关管都关闭时，才会出现高阻状态。当 P0 口用于一般 I/O 口时，内部接 VCC 的那个开关管是与引脚（端口）脱离联系的，这时只有拉地的那个开关管起作用。P0 口作为输出，是必须外接上拉电阻的，否则无法输出高电平；如果 P0 口作为输入，则必须先对端口写 1，使拉地的开关管断开，这时如果不接上拉电阻，则是高阻状态，就是一个双向口，如果接上拉电阻，则本身输出高电平，对输入信号的逻辑无影响。

双向与准双向，根本区别是双向包含高阻这个状态，而不在于是否需要先写 1 或者不写。P1～P3 口因为有内部上拉电阻，所以不是双向；P0 口内部无上拉电阻，在处于数据/地址功能时，自动完成三态的转换，是双向；处于一般 I/O 口时，如果不接外部上拉电阻，而且先向端口写 1，那么就处于高阻状态，此时，它也是一个人为的双向口，这与处于地址/数据功能时的自动双向有区别，也和 P1～P3 处于输入时输出锁存器为 1 是有区别的。当作为输入使用时，将开关断开，这样就只剩下上拉（或者下拉）电阻，因而阻抗比较高，可以由其他设备驱动该 I/O 口。准双向口在作为输入使用时，实际上还是一种输出状态，只是该输出状态的内阻比较大而已。而真正的双向 I/O 口，有方向控制寄存器，作为输入使用时输出部分被断开。

▷▷▷ 7.4.3　单片机端口驱动能力详解

单片机输出低电平时，将允许外部元器件向单片机引脚内灌入电流，这个电流称为“灌电流”，外部电路称为“灌电流负载”；单片机输出高电平时，则允许外部元器件从单片机的引脚拉出电流，这个电流称为“拉电流”，外部电路称为“拉电流负载”。这些电流一般是多少？最大限度是多少？这就是常见的单片机输出驱动能力的问题。早期的 51 系列单片机的带负载能力是很小的，P1～P3 口，每个引脚可以带动 3 个 TTL 输入端，只有 P0 口的能力强，可以带动 8 个。但分析一下 TTL 的输入特性就可以发现，51 单片机基本上没有驱动能力，它的引脚甚至不能带动 LED 进行正常发光。直到 AT89C51 单片机流行起来之后，单片机引脚的能力大为增强，可以直接带动 LED 发光了。

如图 7-2 所示，图中的 D_1、D_2 就可以不经其他驱动器件，直接由单片机的引脚控制发光显示。

从 AT89C51 单片机的 PDF 手册文件中可以看到，稳态输出时，“灌电流”的上限为：

Maximum IOL per port pin: 10mA;
Maximum IOL per 8-bit port 0: 26mA,Ports 1, 2, 3: 15mA;
Maximum total I for all output pins: 71mA.

即每个单个的引脚，输出低电平时，允许外部电路向引脚灌入的最大电流为 10mA；每个 8 位的接口（P1、P2 以及 P3），允许向引脚灌入的总电流最大为 15mA，而 P0 的能力强一些，允许向引脚灌入的最大总电流为 26mA；全部的四个接口所允许的灌电流之和最大为 71mA。而当这些引脚“输出高电平”时，单片机的“拉电流”能力竟然不到 1mA。因此，单片机输出低电平时，驱动能力尚可，而输出高电平时，就没有输出电流的能力。

51 单片机的这些特性，是源于引脚的内部结构，引脚内部结构图详见参考文献[2]。在芯片的内部，引脚和地之间，有个 MOS 管，所以引脚具有下拉的能力，输出低电平时，允许灌入 10mA 的电流；而引脚和正电源之间，有个几百千欧的“内部上拉电阻”，所以引脚在高电平时，能够输出的拉电流很小。但 P0 口的内部没有上拉电阻，所以 P0 口没有高电平输出电流的能力。

如图 7-2 所示，图中的 D_1 是接在正电源和引脚之间的，属于灌电流负载，D_1 在单片机输出低电平时发光。这个发光的电流可以用电阻控制在 10mA 之内。图中的 D_2 是接在引脚和地之间的，属于拉电流负载，D_2 应该在单片机输出高电平时发光。但是单片机此时几乎没有输出能力，必须采用外接“上拉电阻”的方法来提供 D_2 所需的电流。

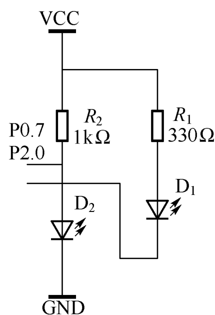


图 7-2 51 单片机的 I/O 端口

从图 7-2 中可以看到，D_2 发光是由上拉电阻 R_2 提供的电流，D_2 导通发光的电压约为 2V，那么发光的电流就是(5-2)/1k，约为 3mA。而当单片机输出低电平（0V），D_2 不发光时，R_2 这个上拉电阻两端的电压，比 LED 发光时还高，达到 5V，其中的电流是 5mA。LED 不发光时，上拉电阻给出了更大的电流，并且这个大于正常发光的电流全部灌入单片机的引脚。如果在一个 8 位的接口，安装了 8 个 1kΩ的上拉电阻，当单片机都输出低电平时，就有 40mA 的电流灌入这个 8 位的接口，如果四个 8 位接口，都加上 1kΩ的上拉电阻，最大有可能出现 32×5=160mA 的电流，都流入到单片机中，这个数值已经超过了单片机手册上给出的上限。如果此时单片机工作不稳定，就是理所当然的了。而且这些电流都是在负载处于无效的状态下出现的，它们都是完全没有用处的电流，只是产生发热、耗电大、电池消耗快等后果。那么，把上拉电阻加大些，可以吗？显然不行，因为需要它为拉电流负载提供电流。对于 LED，如果加大电阻，将使电流过小，发光暗淡，就失去发光二极管的作用了。

对于 D_1，是灌电流负载，单片机输出低电平时，R_1、D_1 通路上会有灌电流；输出高电平时，那就什么电流都没有，此时不产生额外的耗电。

综上所述，“灌电流负载”是合理的；而“拉电流负载”和“上拉电阻”会产生很大的无效电流，这种电路不合理。

7.4.4 LED 结构及发光原理

LED（Light Emitting Diode，发光二极管）的基本结构是一块电致发光的半导体材料，置于一个有引线的架子上，然后四周用环氧树脂密封，起到保护内部芯线的作用，所以 LED 的抗震性能好。LED 结构如图 7-3 所示。

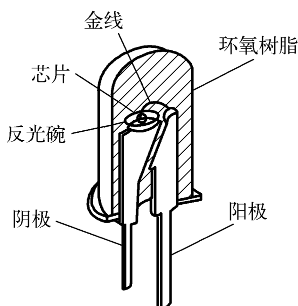


图 7-3 LED 的结构

发光二极管的核心部分是由 P 型半导体和N型半导体组成的晶片，在 P 型半导体和N型半导体之间有一个过渡层，称为 PN 结。在某些半导体材料的 PN 结中，注入的少数载流子与多数载流子复合时会把多余的能量以光的形式释放出来，从而把电能直接转换为光能。PN 结加反向电压，少数载流子难以注入，故不发光。这种利用注入式电致发光原理制作的二极管叫发光二极管，通称 LED。当它处于正向工作状态时（即两端加上正向电压），电流从 LED 阳极流向阴极时，半导体晶体就发出从紫外到红外不同颜色的光线，光的强弱与电流大小有关。

1．LED 光源的特点

1）电压：LED 使用低压电源，供电电压为 6～24V，根据产品不同而异，所以它是一个比使用高压电源更安全的电源，特别适用于公共场所。

2）效能：消耗能量是同光效的白炽灯的 80%。

3）适用性：其体积很小，每个单元 LED 小片是 3～5mm^2 的正方形，所以可以制备成各种形状的器件，并且适合于易变的环境。

4）稳定性：连续工作 10 万个小时，光衰减为初始光强的 50%。

5）响应时间：白炽灯的响应时间为 ms 级，LED 灯的响应时间为 ns 级。

6）对环境污染：无有害金属汞。

7）颜色：改变电流大小可以变色，发光二极管可方便地通过化学修饰方法，调整材料的能带结构和带隙，实现红、黄、绿、蓝、橙多色发光。如小电流时为红色的 LED，随着电流的增加，可以依次变为橙色、黄色，最后为绿色。

8）价格：与白炽灯比较，相同亮度可能需要 300～500 个 LED 才能提供，而几个 LED 的价格就与一只白炽灯的价格相当，因此过去 LED 的价格比较昂贵。但近年来，随着科技的发展，LED 越来越便宜了。



2．LED 驱动

由于单只 LED 管的工作电压低（1.5～2V），个别需达到 4V，同时工作电流仅为 1～5mA，因此可以用 51 单片机的 I/O 端口直接控制。

案例8

 输出模拟量的 I/O 端口

8.1 案例任务

将定时器 0 溢出定为 1/1200s。每 10 次脉冲输出一个 120Hz 频率。这每 10 次脉冲再用来控制高低电平的 10 个比值。这样，在每个 1/120s 的方波周期中，都可以改变方波的输出占空比，从而控制 LED 灯的 10 个级别的亮度。

为什么输出方波的频率要达到 120Hz 这么高？因为如果频率太低，人眼就会看到闪烁感觉。一般要在 60Hz 以上才感觉稍微舒适，120Hz 就基本上看不到闪烁，只能看到亮度的变化。

8.2 案例要点

（1）单片机 I/O 端口的相关知识（参考 7.2 节）

（2）发光二极管的相关知识（参考 7.2 节）

（3）PWM、占空比、平均电压定义

（4）利用 51 单片机的定时计数器产生 PWM 控制信号的方法

8.3 案例设计

8.3.1 硬件电路

输出模拟量的 I/O 端口硬件电路如图 8-1 所示。

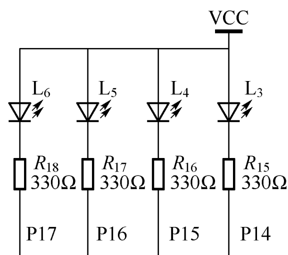


图 8-1 输出模拟量的 I/O 端口硬件电路

8.3.2 软件代码

（1）汇编程序



```
LED      BIT     P1.4
TWTH     EQU     20H            ;周期宽度
TRATIO   EQU     21H            ;子周期占空宽度入口
         ORG     0000H
         LJMP    MAIN
         ORG     000BH
         LJMP    PWM
         ORG     0030H
MAIN:    MOV     TMOD,#01H
         MOV     TH0,#0FCH
         MOV     TL0,#0BFH
         MOV     TRATIO,#00H    ;子周期宽设置
         MOV     TWTH,#0AH      ;母周期宽设置
         SETB    ET0
         SETB    TR0
         SETB    EA
         SETB    LED
LOOP:    INC     TRATIO
         MOV     A,TRATIO
         CJNE    A,#0AH,NEXT    ;不相等则跳转
         MOV     TRATIO,#00H
NEXT:    CALL    DELAY
         JMP     LOOP
         ORG     0200H
PWM:     MOV     TH0,#0FCH
         MOV     TL0,#0BFH
         INC     TWTH
         MOV     A,TWTH
         CJNE    A,#0AH,COMP    ;不相等则跳转
         MOV     TWTH,#00H
         MOV     A,TRATIO
         JZ      COMP           ;A 为 0 则转移
         CLR     LED
COMP:    MOV     A,TRATIO
         CJNE    A,TWTH,LB      ;不相等则跳转
         SETB    LED
LB:      RETI
DELAY:   MOV     R5,#05H
D1:      MOV     R6,#0C8H       ;延时 100ms
D2:      MOV     R7,#0A6H       ;延时 500μs
         DJNZ    R7,$
         DJNZ    R6,D2
         DJNZ    R5,D1
```

```
RET
END
```

（2）C 语言程序

```
#include<reg51.h>
#include<absacc.h>
#include<intrins.h>
#define uchar unsigned char
#define uint unsigned int
sbit H4=P1^4;
/*************精确延时函数*************************************/
//延时(14+6*i)µs     i 最小为 0，最大为 255，即 1544µs
void Delayus(uchar i)
{
        while(i--);
}
void delayms(uint time)
{
        while(time--)
                Delayus(165);
}
//1/1200s 定时器 0 中断
uchar scale;                        //用于保存占空比的输出 0 的时间份额,总共 10 份
void Init_time0(void)
{
        TMOD=0x01;
        TH0 =0xfc;                  //赋 T0 的重装初值，溢出 1 次是 1/1200s
        TL0 =0xbf;
        ET0=1;                      //打开定时器 0 中断
        TR0=1;                      //启动定时器
}
void timer0(void) interrupt 1
{
        static char mult;           //mult 用来保存当前时间在一秒中的比例位置
        mult++;
        TH0 =0xfc;                  //赋 T0 的预置值，溢出 1 次是 1/1200s
        TL0 =0xbf;
        if(mult==10)                //每 1/120s 整开始输出低电平
        {
                mult=0;
                if(scale!=0)        //消除灭灯状态产生的鬼影
                {
                        H4=0;
                }
        }
        if(scale==mult)             //按照当前占空比切换输出高电平
        {
                H4=1;
```

```
        }
}
//模拟 PWM 输出控制灯的 10 个亮度级别
void main(void)                     //  主程序
{
        Init_time0();
        EA=1;                       //打开总中断
        while(1)                    //程序循环
        {
                delayms(500);       //每过一段时间就自动加一个档次的亮度
                scale++;
                if(scale==10)
                        scale=0;
        }
}
```

8.4　案例笔记：51PWM 与平均电压

如图 8-2 所示，电池电压为 10V，如果闭合开关 50ms，在这 50ms 内电灯的电压为 10V，接着断开开关 50ms，在这 50ms 内电灯的电压为 0V。显然，在这 100ms 内，电灯的平均电压为

$$\frac{10\times50+0\times50}{100}\text{V}=5\text{V}$$

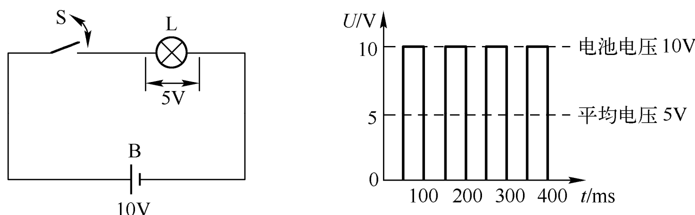


图 8-2　PWM 图示

通过使开关闭合与断开的时间各为 50%，可使电灯亮度降低，从时间平均的角度看，电灯的供电电压变成了 5V。如果使开关闭合与断开的时间分配成 60%和 40%，在 t_a 时间段内，开关闭合的时间为 t_a×60%，而断开的时间为 t_a×40%，所以平均电压为 6V。此时电灯比 10V 时暗，但是比 5V 时稍亮。利用开关对通、断时间的控制来改变平均电压的方法称为 PWM（Pulse Width Modulation）。PWM 信号中高电平在一个周期中所占的比例称为占空比。

PWM 信号是一个数字信号，这是因为在某一时刻，直流电平要么出现，要么不出现，电源以一系列脉冲的形式向负载供电。在带宽足够的情况下，任何模拟信号平均电压都可由 PWM 信号产生。

本案例中发光二极管的亮度变化即是通过 PWM 控制的；亮度的变化与平均电压大小有关。

案例 9

 蜂 鸣 器

▷▷ 9.1 案例任务

通过改变声音的频率和停顿时间的长短就可以产生不同的声音效果，如本案例中的铃声和报警声。本案例中产生方波的频率和发出该频率方波的持续时间用延时程序和定时计数器两种方法实现。

▷▷ 9.2 案例要点

（1）单片机 I/O 端口的相关知识（参考 7.2 节）

（2）蜂鸣器的结构、发声原理及程序驱动

（3）NPN 和 PNP 晶体管的结构、原理

1）PN 结是现代半导体器件的基础。一个 PN 结可制成一个二极管，两个 PN 结即可形成双极型晶体管。

2）双极型半导体晶体管的结构示意图如图 9-1 所示。它有两种类型：NPN 型和 PNP 型。中间部分称为基区，相连电极称为基极，用 B 或 b 表示（Base）。

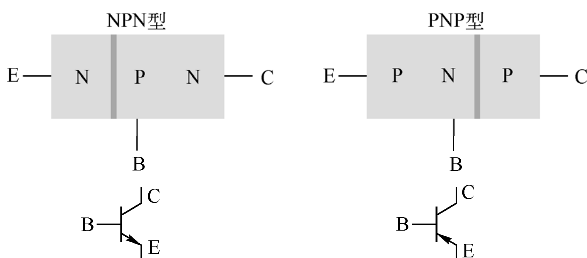


图 9-1 两种极性的双极型晶体管

一侧称为发射区，相连电极称为发射极，用 E 或 e 表示（Emitter）；另一侧称为集电区和集电极，用 C 或 c 表示（Collector）。

E-B 间的 PN 结称为发射结（Je），C-B 间的 PN 结称为集电结（Jc）。

3）半导体晶体管的参数分为直流参数、交流参数和极限参数三大类。

4）半导体晶体管是一种电流控制器件，即通过基极电流或射级电流来控制集电极电流。所谓放大作用，实质上是一种控制作用。需要说明的是，管子的发射结必须正向偏置，而集电结必须反向偏置。

5）晶体管的特性和二极管的是一样的。
（4）单片机 I/O 端口驱动晶体管的常用电路

9.3 案例设计

9.3.1 硬件电路

蜂鸣器电路如图 9–2 所示。

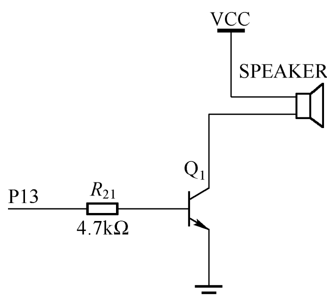


图 9–2 蜂鸣器电路

9.3.2 软件代码

实现各功能程序如下：
（1）铃声

```
        ORG     0000H
        JMP     START
        ORG     0030H
START:  MOV     R4,#20          ;欲鸣叫 1s 需要(25ms+25ms)*20=1s
RING:   MOV     R6,#104         ;产生 320Hz 的声音经计算得 104
        MOV     R5,#8           ;共需要执行 8 周
        ACALL   SOUND           ;发出 320Hz 的声音 25ms
        MOV     R6,#69          ;产生 480Hz 的声音经计算得 69
        MOV     R5,#12          ;共需要执行 12 周
        ACALL   SOUND           ;发出 480Hz 声音 25ms
        DJNZ    R4,RING         ;需要交替鸣叫 1s
        ACALL   D2S             ;静音 2s
        AJMP    START           ;重复执行程序
SOUND:  CLR     P1.3            ;发声子程序
        ACALL   DELAY
        SETB    P1.3
        ACALL   DELAY
        DJNZ    R5,SOUND
        RET
DELAY:  MOV     B,R6
```

```
DL:         MOV     R7,#6
            DJNZ    R7,$
            DJNZ    R6,DL
            MOV     R6,B
            RET
D2S:        MOV     R5,#20
DL1:        MOV     R6,#250
DL2:        MOV     R7,#200
DL3:        DJNZ    R7,DL3
            DJNZ    R6,DL2
            DJNZ    R5,DL1
            RET
            END
```

（2）报警声

```
SPK         BIT     P1.3
FRQ         EQU     1
TMP         EQU     1
STACK       EQU     20
            ORG     00000H
            LJMP    MAIN
            ORG     0000BH
            LJMP    TIMER0
MAIN:       MOV     SP,#(STACK-1)
            MOV     TMOD,#01H
            CLR     A
            MOV     FRQ,A
            MOV     TH0,A
            MOV     TL0,#0FFH
            SETB    TR0
            MOV     IE,#082H
MAIN_LP:    INC     FRQ
            MOV     R7,#04
            LCALL   DELAYMS
            SJMP    MAIN_LP          ; End of main
TIMER0:     MOV     TH0,#0FEH
            MOV     TL0, FRQ
            CPL     SPK
            RETI                     ; End of TIMER0
DELAYMS:    MOV     A,R7
            JZ      END_DLYMS
DLY_LP1:    MOV     R6,#185
DLY_LP2:    NOP
            NOP
            NOP
            DJNZ    R6,DLY_LP2
```

```
            DJNZ    R7,DLY_LP1
END_DLYMS:  RET
            END
```

9.4 案例笔记

9.4.1 51 蜂鸣器结构及发声原理

蜂鸣器是一种一体化结构的电子器件，采用直流或者交流供电，作为发声器件广泛应用于计算机、打印机、复印机、报警器、电子玩具、汽车电子设备、电话机、定时器等电子产品中。蜂鸣器在电路中用字母“H”或“HA”（旧标准用“FM”、“LB”、“JD”等）表示。蜂鸣器的外观如图 9-3 所示。



图 9-3 蜂鸣器外观

虽然单片机对蜂鸣器的控制和对案例 8 中发光二极管的控制是一样的，但硬件电路却有所不同，可参考图 9-2 中蜂鸣器电路。因为蜂鸣器是感性负载，一般不用单片机的 I/O 口直接对其进行操作，而是加一只驱动晶体管。晶体管起开关作用，其基极的高电平使晶体管饱和导通，蜂鸣器发声；而基极低电平则使晶体管关闭，蜂鸣器停止发声。在要求较高的场合，还要加上一只反相保护二极管。

只要让蜂鸣器通过会产生大小变化的电流（即脉冲电流），就能使蜂鸣器发出声音。因此若将程序不断地输出 1→0→1→0→…就可使蜂鸣器发出声音，如图 9-4 所示。

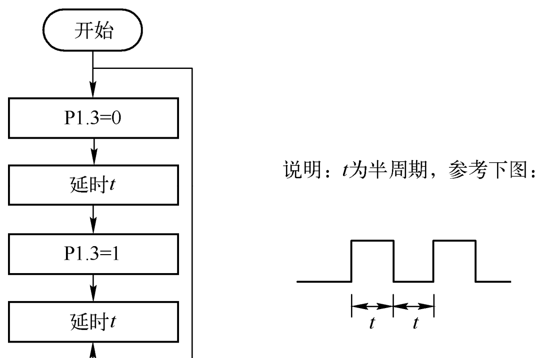
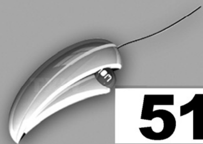


图 9-4 蜂鸣器发声原理



▷▷▷ 9.4.2　8051 晶体管驱动负载的技巧

图 9-5 是 NPN 型、PNP 型晶体管驱动各种负载的典型电路。要求使负载上得到最大的功率，晶体管上消耗最小的功率。使用晶体管驱动负载主要利用晶体管的开关特性，也就是通过控制晶体管在饱和区和截止区之间切换来控制负载的接通和关闭。

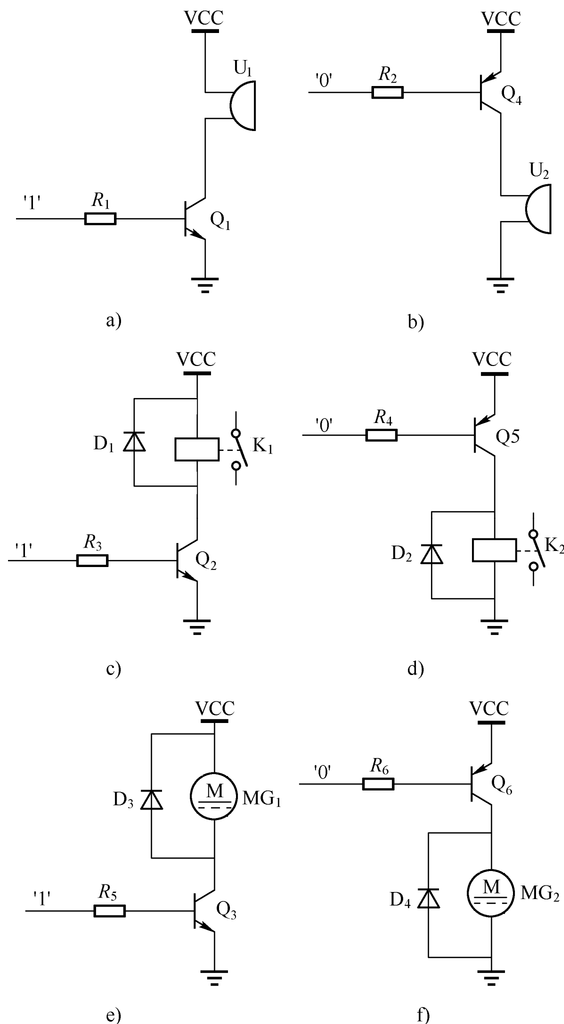


图 9-5　开关晶体管驱动外设

选用 NPN 型还是 PNP 型晶体管驱动负载，要看设计者的要求。如图 9-5a、c、e 是使用 NPN 型晶体管驱动负载的电路，高电平“1”可以控制晶体管导通（负载通电），低电平“0”使晶体管截止（负载断电）。而图 9-5b、d、f 是使用 PNP 型晶体管驱动负载的电路，导通条件刚好相反。

NPN 型晶体管 8050、9013、2N5551 的集电极最大电流分别为 1500mA、500mA、600mA，PNP 型晶体管 8550、9012、2N5401 的集电极最大电流分别为 1500mA、500mA、500mA。驱动蜂鸣器、继电器、电动机等负载，主要看晶体管集电极电流是否能满足负载要

求。图 9-5 所示的各种驱动电路中，经常取基极电阻约 1kΩ，对应的基极电流为 4~5mA，集电极电流能满足负载要求。

▷▷▷ 9.4.3 开关晶体管使用误区

在数字电路设计中，往往需要把数字信号经过开关扩流器件来驱动一些蜂鸣器、LED、继电器等需要较大电流的器件，用得最多的开关扩流器件要数晶体管。然而在使用的过程中，如果电路设计不当，晶体管无法工作在正常的开关状态，就达不到预期的目的，有时就是因为这些小小的错误而导致重新制作 PCB，导致浪费。下面来看几个晶体管作开关的常用电路画法。这几个例子都是蜂鸣器作为被驱动器件。

图 9-6a 电路用的是 NPN 管，注意蜂鸣器接在晶体管的集电极，驱动信号可以是常见 3.3V 或者 5V 电平的 TTL 信号，高电平开通，电阻按照经验法可以取 4.7kΩ，开通时假设为高电平 5V，基极电流 I_b=(5−0.7)/4.7k=0.9mA，可以使晶体管完全饱和。图 9-6b 电路用的是 PNP 管，同样把蜂鸣器接在晶体管的集电极，不同的是驱动信号是 5V 的 TTL 电平。以上两个电路都可以正常工作，只要 PWM 驱动信号工作在合适的频率，蜂鸣器（有源）都会发出最大的声音。

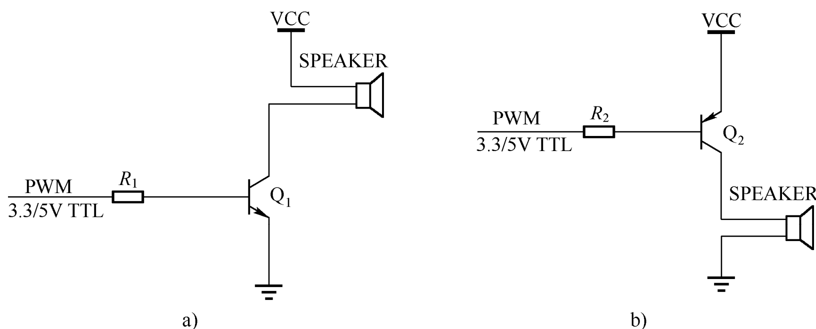


图 9-6 开关晶体管图 1

图 9-7 的这两个电路相比图 9-6 来说，最大的区别在于被驱动器件接在晶体管的发射极。同样对于图 9-7a 电路，开通时假设为高电平 5V，基极电流 I_b=(5−0.7−U_L)/4.7k，其中 U_L 为被驱动器件上的压降。可以看到，同样取基极电阻为 4.7kΩ，流过的基极电流会比图 9-6a 电路要小，小多少要看 U_L 是多少。如果 U_L 比较大，那么相应的 I_b 就小，很可能导致晶体管无法工作在饱和状态，使得被驱动器件无法动作。有人会说把基极电阻减小就可以了，可是被驱动器件的压降是很难获知的，有些被驱动器件的压降是变动的，这样一来基极电阻就较难选择合适的值，阻值选择太大就会驱动失败，选择太小，损耗又变大。所以一般情况下，不建议选用图 9-7 的这两种电路。

图 9-8 中驱动信号为 3.3V TTL 电平，而被驱动器件开通电压需要 5V。在 3.3V 的电路中，很容易就设计出这两种电路，而这两种电路都是错误的。先分析图 9-8a 电路，这是典型的“发射极正偏，集电极反偏”的放大电路，或者叫射极输出器。当 PWM 信号为 3.3V 时，晶体管发射极电压为 3.3−0.7V＝2.6V，无法达到期望的 5V。图 9-8b 电路也是一个很失败的电路，首先这个电路开通是没有问题的，当驱动信号为低电平时，被驱动器件可以正常动作。然而这个电路是无法关断的，当驱动信号 PWM 为 3.3V 高电平时，U_{be}＝5－3.3V＝1.7V，仍然可以使晶体管开通，但是无法关断。有人会认为这个电路没有问题，而且单片机的电压也是 3.3V。但如果

用的是 OD（开漏）驱动方式，而且是真正的 OD 或者是 5V 容忍的 OD，例如 STM32 的很多 I/O 口都可以设置为 5V 容忍的 OD 驱动方式（但是有些是不行的）。当驱动信号为 OD 门驱动方式时，输出高电平，信号就变成了高阻态，流过基极的电流为零，晶体管可以有效关断，这时图 9-8b 电路依然有效。

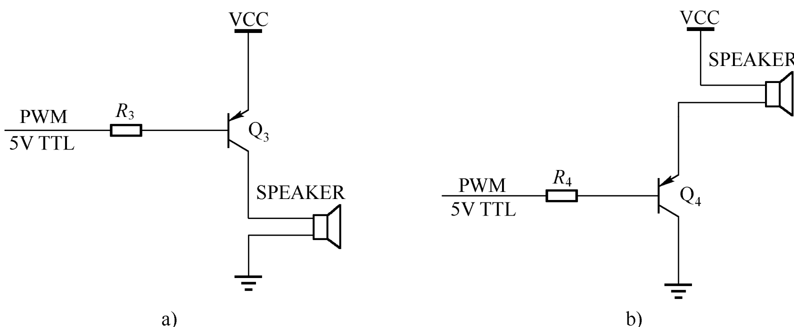


图 9-7　开关晶体管图 2

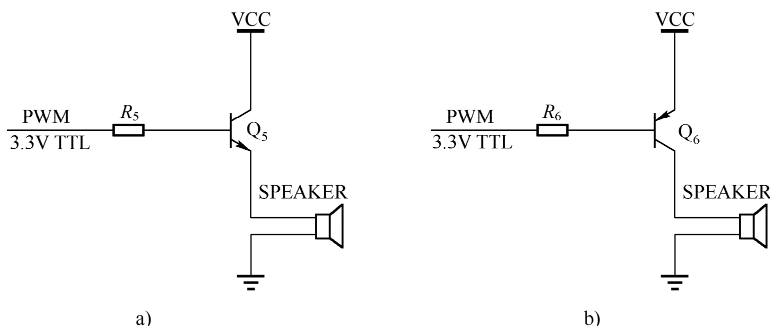


图 9-8　开关晶体管图 3

综合以上几种电路的情况分析，得到图 9-9 这两个较优的驱动电路，与图 9-6 不同的是，图 9-9 在基极与发射极之间多加了一个 100kΩ的电阻，这个电阻可以使晶体管有一个已知的默认状态。当输入信号去除时，晶体管还处于关断状态。从安全和稳定的方面考虑，多加的这个电阻还是很有必要的，或者说可以让晶体管工作在更好的开关状态。

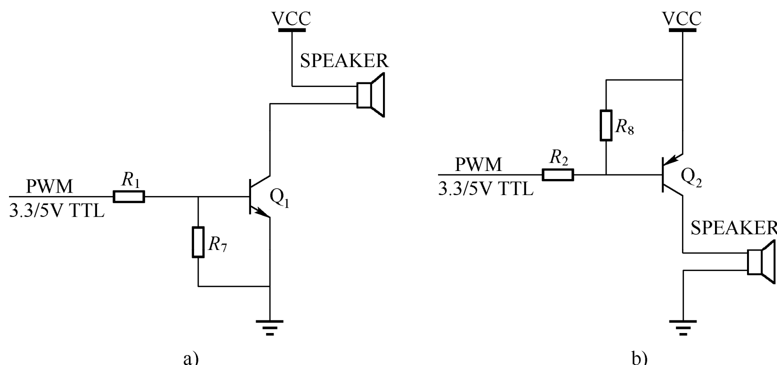


图 9-9　开关晶体管图 4

晶体管作为开关器件，虽然驱动电路很简单，但要使电路工作更加稳定可靠，还是不能掉以轻心。为了不出错，建议优先采用图 9-9 的电路，尽量不采用图 9-7 的电路，避免使用图 9-8 的工作状况。

案例10



并口的扩展



10.1 案例任务

8255 扩展口所接发光二极管 L7 点亮，其他灯灭，延时一段时间，然后 L8 灯亮 L7 灯灭，重复执行下去，直到 L14 灯灭 L7 灯亮，循环执行。

10.2 案例要点

（1）接口的概念、接口的基本功能

（2）I/O 信号的类型

（3）数据端口、状态端口、控制端口

（4）I/O 端口的编址方法

（5）CPU 与外设数据传送的四种方式

即无条件传送、查询式传送、中断式传送和 DMA 传送的特点及相互比较；无条件传送的电路及编程；查询式传送的电路及编程。

（6）8255A 的内部结构与引脚功能

1）三个输入/输出端口：PortA、PortB 和 PortC（简称 PA、PB 和 PC）。每一个端口都是 8 位，都可以选择作为输入或输出，但功能上有着不同的特点。

2）A 组和 B 组控制电路。

3）数据总线缓冲器。

4）读/写和控制逻辑。

（7）掌握 8255 方式选择控制字各位的意义

1）方式命令是对 8255 的三个端口的工作方式及功能进行指定，即进行初始化，初始化工作要在主程序开始时完成。

2）按位置位/复位命令只是对 PC 口的输出进行控制，使用它不会破坏已经建立的三种工作方式，并且是对它们实现动态控制的一种支持。它可放在初始化程序以后的任何地方。

3）两个命令的最高位（D7）作特征位，设置特征位的原因是为了识别两个不同的命令。

4）按位置位/复位的命令代码只能写入命令口。

5）PA 口、PB 口也可以按位输出高低电平，但是，它与前面的按位置位/复位命令有本质的差别，并且实现的方法也不同。PA 口、PB 口按位输出实现的具体做法：若要使某一位置高电平，则先对端口进行读操作，将读入的原输出值，“或”上一个字节，字节中使该位

为 1，其他位为 0，然后再送到同一端口，即可使该位为 0，然后再送到同一端口，即可使该位置位。

（8）8255 方式 0 的工作特点

1）方式 0 是一种基本 I/O 工作方式。通常不用联络信号，或不使用固定的联络信号。方式 0 可以使用在无条件传送和查询传送两种场合。

2）在方式 0 下，彼此独立的两个 8 位和两个 4 位并行口，都能被指定作为输入或者输出用。

3）在方式 0 下不设置专用联络信号线，需要联络时，可由用户任意指定 PC 口中的某根线完成某种联络功能，这与后面要讨论的在方式 1、方式 2 下设置固定的专用联络信号线不同。

4）是单向 I/O，一次初始化只能指定端口（PA、PB 和 PC）作输入或输出，不能指定端口既作输入又作输出。

5）输出可被锁存，输入不能被锁存。

（9）8255 方式 1 的工作特点

1）状态字是在 8255 的 I/O 操作过程中由内部产生，从 PC 口读取的，因此从 PC 口读出的状态字是独立于 PC 口的外部引脚的，或者说与 PC 口的外部引脚无关。

2）状态字中供微处理器查询的状态位有输入时 IBF 位和 INTR 位、输出 OBF 位和 INTR 位。

3）状态字中的 INTE 位是控制标志位，控制 8255 能否提出中断请求，因此它不是 I/O 操作过程中自动产生的状态，而是由程序通过按位置位/复位命令来设置或清除的。

（10）8255 方式 2 的工作特点

1）双向总线方式。方式 2 相当于方式 1 的输入和输出组合。只有 PA 口具备双向总线方式。PA 口为双向选通 I/O 或叫双向应答式 I/O。一次初始化可指定 PA 口既作输入口又作输出口。设置专用的联络信号线和中断请求信号线，因此，方式 2 下可采用中断方式和查询方式与微处理器交换数据。

2）PA 口可以工作于方式 2，此时 PC 口有 5 条线固定为 PA 口和外设之间的联络信号线。PC 口其他 3 条线可以作为 PB 口方式 1 下的联络线，也可以和 PB 口一起成为方式 0 的 I/O 线。

3）各联络线的定义及其时序关系和状态是在方式 1 下输入和输出两种操作的组合。

4）数据在输入输出时都被锁存。

（11）了解其工作时序

了解 8255 方式 1 和方式 2 的工作特点及工作时序；掌握 8255 按位置位/复位操作控制字的设置方法。

（12）8255 与单片机的接口方法及端口寻址

8255 中有 3 个输入/输出端口，另外，内部还有一个控制字寄存器，共有 4 个端口，要有 2 个输入端来加以选择，这两个输入端通常连到地址总线的最低两位 A1 和 A0。

（13）对 8255 进行初始化编程

10.3 案例设计

10.3.1 硬件电路

并口的扩展电路如图 10-1 所示。

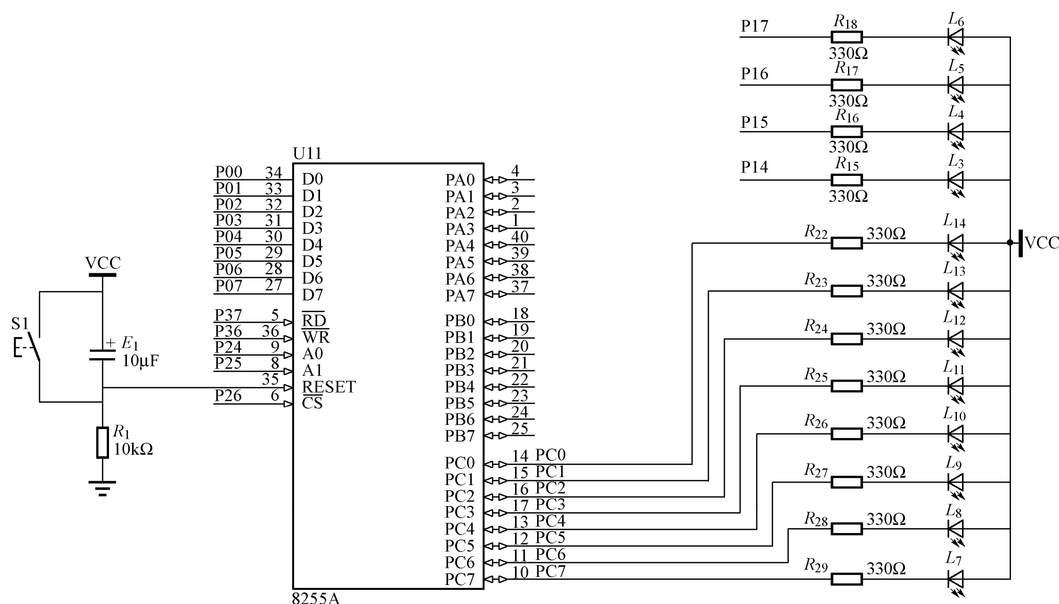


图 10-1　并口的扩展电路

10.3.2 软件代码

（1）汇编程序

```
PORTA      EQU     8FFFH          ;8255PA 口地址
PORTB      EQU     9FFFH          ;8255PB 口地址
PORTC      EQU     0AFFFH         ;8255PC 口地址
CADDR      EQU     0BFFFH         ;8255 控制字地址
           ORG     0000H
           JMP     MAIN
           ORG     0030H
MAIN:      MOV     A,#80H         ;方式 0
           MOV     DPTR,#CADDR
           MOVX    @DPTR,A        ;设置 8255 工作方式
           SETB    P2.6
LOOP:      MOV     A,#0FEH        ;设置显示码
           MOV     R2,#8          ;设置计数值
OUTPUT:    MOV     DPTR,#PORTC
           MOVX    @DPTR,A        ;显示码送 PA 口显示
```

```
                CALL        DELAY
                RL          A                   ;显示码数据移位
                DJNZ        R2,OUTPUT
                LJMP        LOOP
DELAY:          MOV         R6,#0               ;延时子程序
                MOV         R7,#0
DELAYLOOP:      DJNZ        R6,DELAYLOOP
                DJNZ        R7,DELAYLOOP
                RET
                END
```

（2）C 语言程序

```
unsigned char xdata porta _at_ 0x8fff;
unsigned char xdata portb _at_ 0x9fff;
unsigned char xdata portc _at_ 0xafff;
unsigned char xdata cadrr _at_ 0xbfff;
unsigned char xdata *i _at_ 0x0060;
unsigned char xdata *b _at_ 0x0070;      //必须为字符型且作用域必须相同
unsigned char n;
unsigned int a;
unsigned char n1=0;
unsigned char m=0x7f;
#include<AT89X51.h>
#include<INTRINS.H>
void main (void)
{
 i = &cadrr;
 *i=0x80;
 b=&portc;
 while (1)
      {
       for ( n=0;n<8;n++)
            {
            n1++;
              *b=m;
            for (a=0;a<60000;a++) ;
              m=_cror_ (m,1);
            }
      }
}
```

10.4 案例笔记

10.4.1 8255 各工作方式总结

8255 各工作方式端口总结见表 10-1。

表 10-1　8255 各工作方式端口总结

端　口		方式 0	方式 1		方式 2
			输入	输出	双向
PC 口	PC0	基本 I/O	INTRB	INTRB	I/O
	PC1		IBFB	$\overline{OBFB}$	I/O
	PC2		$\overline{STBB}$	$\overline{ACKB}$	I/O
	PC3		INTRA	INTRA	INTRA
	PC4	基本 I/O	基本 I/O	I/O	$\overline{STBA}$
	PC5		IBFA	I/O	IBFA
	PC6		I/O	$\overline{ACKA}$	$\overline{ACKA}$
	PC7		I/O	$\overline{OBFA}$	$\overline{OBFA}$
PA 口		基本 I/O	选通 I/O		双向数据传送
PB 口		基本 I/O	选通 I/O		

▷▷▷ 10.4.2　微机扩展 I/O 接口的基础知识

（1）微机与 I/O 之间接口信号

微机与 I/O 接口、外设连接如图 10-2 所示。微机与 I/O 接口之间通过数据总线、地址总线、控制总线相连。I/O 接口与外设交换的数据信息从广义上来讲包括数据信息、状态信息、控制信息。

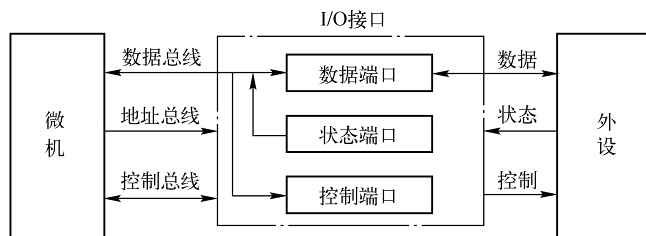


图 10-2　微机与 I/O 接口、外设连接

1）数据信息：微机与外设交换的数据，经接口数据缓冲器传送。

2）状态信息：状态信息是反映外设当前所处的工作状态，以作为微机与外设间可靠交换数据的条件。当输入时，它告诉微机，有关的输入设备是否准备好（Ready=1？）；输出时它告诉微处理器，输出设备是否空闲（Busy=0？）。

3）控制信息：用于控制外设的启动和停止，以及给出命令字，用于设置接口的工作方式。

通常接口中三种信息由不同的寄存器传送，如数据输入寄存器、数据输出寄存器、状态寄存器和控制（命令）寄存器，它们使用不同的端口地址来区分不同性质的信息。所谓串行接口和并行接口，是指外设和接口一侧的传送方式，而在单片机和接口一侧，数据总是并行传送的。

（2）系统扩展的总线连线原则

系统的扩展归结为三总线的连接，连接的方法很简单，连线时应遵守下列原则：

1）数据线连数据线，地址线连地址线，控制线连控制线。针对 51 单片机要特别注意的是，程序存储器接 PSEN，数据存储器接 RD 和 WR。

2）控制线相同的地址线不能相同，地址线相同的控制线不能相同。

3）片选信号有效的芯片才选中工作，当一类芯片仅一片时片选端可接地，当同类芯片多片时片选端可通过线译码、部分译码、全译码接地址线（通常是高位地址线），在单片机中多采用线选法。

（3）51 单片机的总线

51 单片机的总线结构如图 10-3 所示。

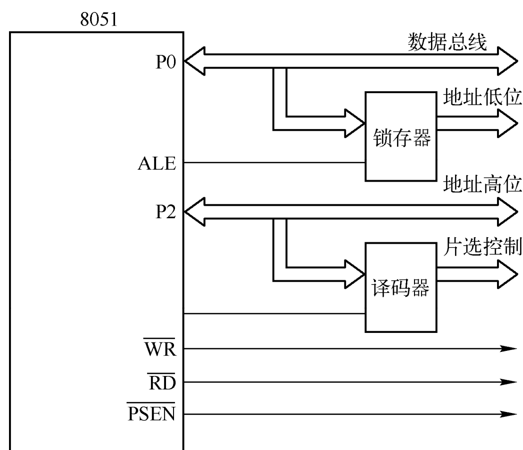


图 10-3　51 单片机的总线结构

51 单片机的控制线功能总结见表 10-2。

表 10-2　51 单片机的控制线

名　称	功　能
$\overline{PSEN}$	片外取指信号（片外程序存储器读）。输出端低电平有效。通过 P0 口读回指令或常数。控制的是片外程序存储器
ALE	地址锁存信号。P0 口是数据/地址复用口；ALE 低电平时，P0 口出现数据信息；ALE 高电平时，P0 口出现地址信息。 用下降沿“↓”锁存 P0 口的低 8 位地址到外部锁存器
$\overline{EA}$ / VPP	程序存储器选择信号 $\overline{EA}$=0，选外部ROM；1，地址小于4k时，选内部ROM；地址大于4k时，选外部ROM

▷▷▷ 10.4.3　I/O 接口的数据传送方式

微机与外设之间进行信息交换有四种方式：程序查询方式、程序中断方式、直接内存访问方式（DMA）和通道方式。

为便于理解上述四种基本交换方式，不妨设想以下情节：

某公司经理掌管 8 个独立部门，现经理想要给每个部门分配三个项目（project），那么他该怎么做呢？

第一种方法：先给部门甲布置一个项目，守候等甲完成这个 project 后，再分配甲第二个 project，仍静候等其完成第二个 project 后，布置第三个 project 给甲，等甲完成第三个 project，再到部门乙布置 project，过程与在部门甲完全相同。就这样，部门乙完成后，到部门丙，直至 8 个部门全部完成 project。显然这种方法效率太低，令人更不能忍受的是各部门在完成布置的 project 过程中，经理一直在守候，什么事也没干。

于是，这个经理想出了第二种方法。给每个部门各先布置一个 project，并约定谁完成 project 后就向他报告，再给报告部门布置第二个 project。看来这种方法提高了工作效率而且在未接到部门完成 project 报告之前，经理还可以在办公室里做其他事。

但这种方法还可以改进，于是他又想出了第三种方法：进行批处理，每个部门各自取走 3 个 project，哪个部门完成他们所负责的 3 个 project 后，再向他报告。显然，这种方法工作效率大大提高了，他可以腾出更多的时间来干其他事了。

还有没有更好的办法呢？他想出了第四种方法，进行权力下放，把布置 project 的事交给他的秘书全权处理，自己只是在必要时才过问一下。

现在，可以回过来想一想：要是我们将这位经理视为微机，而 8 个部门视为 8 个外设，project 视为微机与外设之间交换的信息，那么问题就简单了。第一种方法，对应于程序查询方式；第二种方法，对应于程序中断方式；第三种方法对应于直接内存访问方式（DMA），第四种方法则对应于通道方式。

通道方式中，通道也是一种“接口”，比普通的系统总线具有更强的功能，它的内部一般有单片机微处理器，可执行简单的“通道程序”，是一种为微处理器分担管理 I/O 操作的控制器。

▷▷▷ 10.4.4　可编程芯片总体要求

对于可编程芯片，关键要从芯片的外部接口特性、内部工作原理和应用编程的方法几个方面来掌握：

1）掌握芯片的外部引线及其功能，以便将它连接到微机系统中。

2）掌握芯片的工作方式及工作特点，以便选择适合于用户要求的工作方式。

3）掌握芯片内部的控制字和状态字，它将决定芯片的工作方式及工作特点。

4）掌握芯片的寻址和内部各端口寄存器的读/写控制方法。

5）掌握芯片的初始化编程。一般而言，可编程芯片若不进行初始化，将不能工作。

典型的并行接口通常由如下部分组成：

1）数据端口（接外设）：包括相应的输入缓冲寄存器和输出缓冲寄存器。

2）地址、数据、控制（片选和控制电路）总线：与微处理器通信。

3）控制寄存器（保存控制字）用来接收微处理器对它的控制指令，如选择数据端口、选择端口传送方向（输入输出或双向）、选择与微处理器交换信息的方法（查询或中断）。

4）有一个状态寄存器提供各种状态供微处理器查询。

5）中断电路。

案例11

 方式可控的流水灯

11.1 案例任务

改变 2 位拨码开关，4 种状态使 P1.4～P1.7 所接的 4 个发光二极管发生相应亮灭变化。

11.2 案例要点

（1）单片机 I/O 端口的相关知识（参考 7.2 节）
（2）拨码开关的原理
注意拨码开关拨完键后，手离开开关后开关值不变；而按键手松开后，按键断开。
（3）拨码开关与 51 单片机 I/O 口的硬件连接及驱动程序的编写
1）注意上拉电阻的作用。
2）读并口数据之前有个预备动作：MOV P1,#0FFH，这样才能准确读入 I/O 接口值。
（4）分支程序的编写方法

11.3 案例设计

11.3.1 硬件电路

方式可控的流水灯电路如图 11-1 所示。

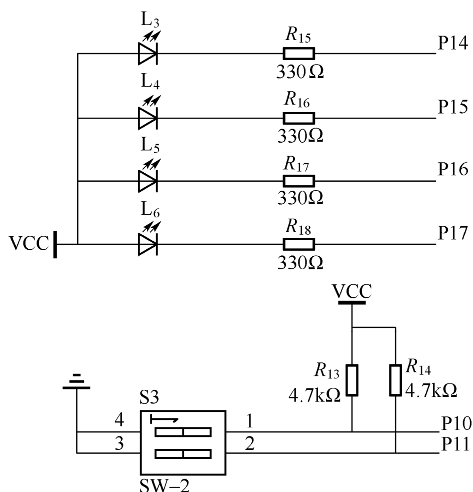


图 11-1 方式可控的流水灯电路

▷▷▷ 11.3.2 软件代码

（1）汇编程序

```
        ORG     0000H
        JMP     MAIN
        ORG     0030H
MAIN:   MOV     P1,#0FFH
        MOV     A,P1
        ANL     A,#03H
        RL      A
        MOV     DPTR,#TAB
        JMP     @A+DPTR         ;采用查表法跳转到相应程序
TAB:    AJMP    FSH1
        AJMP    FSH2
        AJMP    FSH3
        AJMP    FSH4
FSH1:   CLR     P1.4
        SETB    P1.5
        SETB    P1.6
        SETB    P1.7
        JMP     MAIN
FSH2:   SETB    P1.4
        CLR     P1.5
        SETB    P1.6
        SETB    P1.7
        JMP     MAIN
FSH3:   SETB    P1.4
        SETB    P1.5
        CLR     P1.6
        SETB    P1.7
        JMP     MAIN
FSH4:   SETB    P1.4
        SETB    P1.5
        SETB    P1.6
        CLR     P1.7
        JMP     MAIN
        END
```

（2）C 语言程序

```
#include<reg51.h>
#include<absacc.h>
#include<intrins.h>
sbit  H1=P1^4;
sbit  H2=P1^5;
sbit  H3=P1^6;
sbit  H4=P1^7;
void main(void)
```

```
{
char i;
        H1=1;
    H2=1;
    H3=1;
    H4=1;
    while(1)
    {
            i=P1&0x03;
        switch(i)
        {
            case 0:H1=0;
                   H2=1;
                   H3=1;
                   H4=1;
                   break;
            case 1:H1=1;
                   H2=0;
                   H3=1;
                   H4=1;
                   break;
            case 2:H1=1;
                   H2=1;
                   H3=0;
                   H4=1;
                   break;
            default:H1=1;
                    H2=1;
                    H3=1;
                    H4=0;
        }
    }
}
```

11.4 案例笔记：分支结构的常见错误

分支结构的常见错误总结如下：

（1）if 语句的常见错误

1）错将赋值语句用作关系表达式，将 if(x==3) 错写成 if(x=3)（最好的形式是 if(3==x)）。

2）逻辑表达式用错，例如，将 if ((x<=3)&& (x>1)) 错写成 if (3>=x>1)。

3）if 语句的布尔表达式后面加了分号，变成：

```
if(布尔表达式);
```

多了一条空语句，导致程序流程完全错误（无 else 时），或者通不过编译（有 else 分支时）。

4）在 if 语句中的一个嵌入位置嵌入了多条语句，但没有用花括号括起来以构成一条复合语句。



例如：

```
if  (表达式)
语句 1;
语句 2;
else
语句 3;
语句 4;
语句 5;
```

正确的应为

```
if  (表达式)
  {
    语句 1;
    语句 2;
  }
else
  {
    语句 3;
    语句 4;
  }
    语句 5;
```

5）将嵌套关系的 if 语句错用成并列关系的 if 语句，或将并列关系错用成嵌套关系。

6）else 分句与哪个 if 配对的关系搞错。

7）if（y<=6）错写成 if（y< =6）（在 < 和 = 之间多了一个空格），>=、==、和!=都不能分开写。

8）if 语句的布尔表达式整体上没用括号括住，例如，将 if ((x<=1)&& (x>3)) 写成 if(x<=1)&& (x>3)。

9）在应该用有 else 的 if 语句时，用了无 else 的 if 语句，或者正好相反。

10）if 语句的关键字用错，将 if 写成 IF，或将 else 错写成 eles。

（2）switch 语句的常见错误

1）漏写了本来应当有的 break，导致流程错误。

2）在 switch（整型或字符型表达式）右边多加了分号。

3）漏写了 switch 语句本该有的一对花括号。

4）switch 后面的表达式用了实型表达式。

5）在“case 具体取值：”中漏掉了冒号，或错用逗号。

6）在不应该省略 default 分句时，省略了该分句，导致程序的健壮性差。

（3）while 语句的常见错误

1）while 后面的布尔表达式忘了用括号括住，变为 while i<=100。还有 while

(i<=n)&&(j!=0)也少了括号，正确的应为 while ((i<=n)&&(j!=0))或者 while (i<=n&&j!=0)。

2）布尔表达式的后面多了分号，变成 while(布尔表达式);导致循环体是一条空语句，而且循环是无限循环（因为在每次执行循环体后，循环的条件永远不变）。

3）循环前忘记对变量进行初始化，包括循环变量和循环中用到的变量。

4）在循环体中忘记写循环变量的更新语句，导致循环执行的条件（即布尔表达式的值）永远为“真”，产生了无限循环。在循环变量为字符变量时最容易犯此错误。

5）漏写了括住循环体的花括号，导致循环体只有一条语句，这样也常常造成死循环。因为漏写花括号后，对循环变量的更新成为循环后的下一条语句。

6）循环的终止条件发生错误，即循环多了一次或少了一次。比如将 while（循环变量<=常量）错写成 while（循环变量<常量），造成循环体中的语句少执行了一次，导致结果不对。

7）循环体的花括号后多加了一个分号。

8）在循环体中，出现改变循环变量终值取值的语句（如果终值是用一个变量来表示的）。

do-while 语句与 while 语句类似，只不过 do-while 语句的 while(布尔表达式)后面一定要有分号。

无论是 while 循环，还是 do-while 循环，初学者最常犯的两类错误如下：

1）循环前忘记了循环变量的初始化，造成循环次数错。

2）循环体中漏掉了对循环变量进行更新的语句，导致无限循环。

为此高级程序设计语言的设计者们提供了一种更好用的循环语句，即 for 语句。

（4）for 语句的常见错误

for 循环语句的常见错误除了 while 的常见错误外（仅有忘记循环前的初始化和对循环变量更新这两个错误不易犯），还容易犯以下错误。

1）三个表达式之间的两个分号被省略掉了。比如正确的应为

```
for(; i<10 ;)
```

可能会被省略掉前后的任意一个或两个分号。

2）三个表达式之间用逗号隔开。比如：

```
for(i=1,   i<10 , i++)
```

或者

```
for(i=1,j=M, i<M,i++,j--)
```

正确的应为

```
for(i=1,j=M;i<M ;i++,j--)
```

其中逗号表达式 i=1，j=M 是表达式 1，而第 3 个表达式 i++, j--是另一个逗号表达式。

3）表达式 3 后面多加了分号，比如 for（i=0;i<10;i++;）。

案例12



数字显示器



12.1 案例任务

8255 所接的一个 LED 显示器从 0 亮到 9，数码管动态显示。

12.2 案例要点

（1）LED 的结构及工作原理

1）共阳极 LED：8 个发光二极管的阳极都连在一起的，称之为共阳极 LED 显示器。

2）共阴极 LED：8 个发光二极管的阴极都连在一起的，称之为共阴极 LED 显示器。

3）段码和字型的关系。

（2）多位并联的 LED 结构及工作原理

（3）静态显示和动态显示

1）所谓静态显示，就是每一个显示器都要占用单独的具有锁存功能的 I/O 接口用于笔划段字形代码。

2）动态扫描显示接口是单片机中应用最为广泛的显示方式之一。其接口电路是把所有显示器的 8 个笔划段 a～h 同名端连在一起，而每一个 LED 的公共极 COM 是各自独立地受 I/O 线控制。CPU 向字段输出口送出字形码时，所有 LED 接收到相同的字形码，但究竟是哪个 LED 亮，则取决于 COM 端，而这一端是由 I/O 控制的，所以就可以自行决定何时显示哪一位了。而所谓动态扫描就是指采用分时的方法，轮流控制各个 LED 的 COM 端，使各个 LED 轮流点亮。

（4）LED 动态显示的接口电路及动态扫描显示程序的设计方法

（5）LED 与 8255 接口方法及相应的驱动程序设计

12.3 案例设计

12.3.1 硬件电路

数字显示器电路如图 12-1 所示。

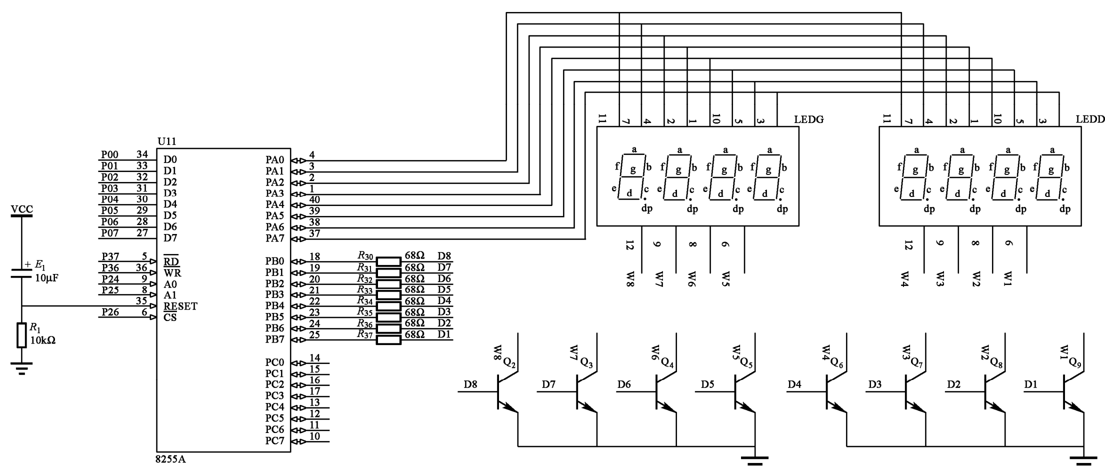


图 12-1　数字显示器电路

▷▷▷ 12.3.2　软件代码

（1）汇编程序

```
PORTA       EQU     8FFFH            ;8255A 口地址
PORTB       EQU     9FFFH            ;8255B 口地址
PORTC       EQU     0AFFFH           ;8255C 口地址
CADDR       EQU     0BFFFH           ;8255 控制字地址
            ORG     0000H
            JMP     MAIN
            ORG     0030H
MAIN:       MOV     A,#80H           ;方式 0
            MOV     DPTR,#CADDR
            MOVX    @DPTR,A          ;设置 8255 工作方式
            MOV     DPTR,#PORTB
            MOV     A,#80H           ;选中右边第一个七段码管
            MOVX    @DPTR ,A
LOOP:       MOV     R0,#00H          ;设置显示数初值
            MOV     r1,#0AH          ;设置循环变量
LOOP1:      MOV     A,R0
            MOV     DPTR ,#TAB
            MOVC    A,@A +DPTR
OUTPUT:     MOV     DPTR,#PORTA
            MOVX    @DPTR,A          ;显示码送 PA 口显示
            CALL    DELAY
            INC     R0               ;指向下一个显示数
            DJNZ    R1,LOOP1
            JMP     LOOP
tab:        DB      3FH,06H,5BH,4FH,66H,6DH,7DH,07H,7FH,6FH
                    ; '0"1"2"3"4"5"6"7"8"9'
DELAY:      MOV     R5,#5
```

```
D3:     MOV     R7,#255
d1:     MOV     R6,#255
d2:     DJNZ    R6,d2
        DJNZ    R7,d1
        DJNZ    R5,D3
        RET
        END
```



（2）C 语言程序

```
unsigned char xdata porta _at_ 0x8fff;
unsigned char xdata portb _at_ 0x9fff;
unsigned char xdata portc _at_ 0xafff;
unsigned char xdata caddr _at_ 0xbfff;
unsigned char xdata *a _at_ 0x0060;
unsigned char xdata *b _at_  0x0070;
unsigned char xdata *c _at_ 0x0075;
#include<AT89X51.h>
unsigned char led[10]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};
void main (void)
{
        unsigned char i;
        unsigned int n;
        a=&caddr;
        *a=0x80;
        b=&portb;
        c=&porta;
        for (i=0;i<10;i++)
            {
             *c=led[i];
             *b=0x80;
             for (n=0;n<30000;n++);
            }
}
```

▷▷ 12.4　案例笔记：51 LED 显示码便于移植的解决方法

51 LED 显示码便于移植的解决方法如下：

1．汇编语言

先定义笔画对应的 bit 位置：

```
........
a equ 01h
b equ 02h
c equ 04h
d equ 08h
```

```
e equ 10h
f equ 20h
g equ 40h
dot equ 80h

.......
;定义显示码如下：
........
    db    a+b+c+d+e+f        ;0
    db    b+c                ;1
    db    a+b+d+e+g          ;2
    db    a+b+c+d+g          ;3
    db    b+c+f+g            ;4
    db    a+c+d+f+g          ;5
    db    a+c+d+e+f+g        ;6
    db    a+b+c              ;7
    db    a+b+c+d+e+f+g      ;8
    db    a+b+c+d+f+g        ;9

........
```

2．C 语言

先定义笔画对应的 bit 位置：

```
.......
#define a      0x01
#define b      0x02
#define c      0x04
#define d      0x08
#define e      0x10
#define f      0x20
#define g      0x40
#define dot    0x80

.......

uchar code dis_code[] = {
        a+b+c+d+e+f,        // 0
        b+c,                // 1
        a+b+d+e+g,          // 2
        a+b+c+d+g,          // 3
        b+c+f+g,            // 4
        a+c+d+f+g,          // 5
        a+c+d+e+f+g,        // 6
```

```
a+b+c,                  // 7
a+b+c+d+e+f+g,          // 8
a+b+c+d+f+g             // 9
};

........
```



每次使用只要修改笔画对应的 bit 位置就可以了。

案例13

 行 列 键 盘

13.1 案例任务

按下键盘上的数字键，LED 显示相应数字。例如按下 0，LED 显示 0。

13.2 案例要点

（1）按键“稳定闭合”及“去抖”的概念（见图 13-1）

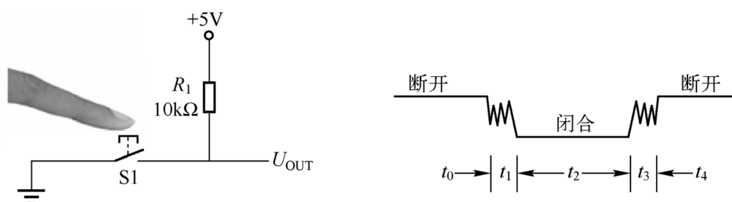


图 13-1 按下开关时手指的抖动及输出端的电平变化

当手指按下开关过程中产生一个跳变电平过程，开关在一个很短的时间内，出现了多次的接通与断开。这样，输出端 U_{OUT} 出现一个电平在+5～0V 之间抖动的过程。这对于单片机这个具有高灵敏引脚的器件来说，会产生误动作。按键去抖有两种方法：

1）硬件去抖（见图 13-2）。首先利用 *RC* 电路将开关的抖动平缓化，然后用带施密特触发器的反相器 74HC14 将信号进行门限化。硬件消抖由于增加硬件开销，通常不采用。

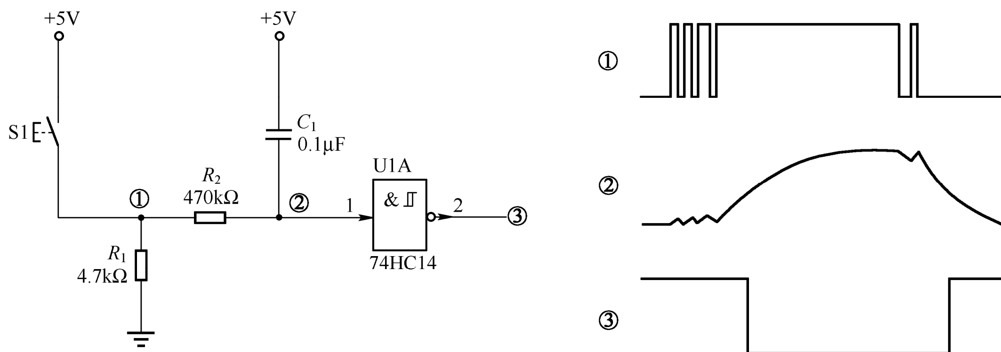


图 13-2 硬件去抖

2）软件去抖。检测到有键按下，软件延时 10ms 后，再次检查该键电平是否仍保持闭合状态，如果保持闭合状态，则确认有键按下；否则重新检查，这样就能消除抖动的影响。详见本案例。

（2）行列式键盘工作原理

（3）掌握行列式键盘与 51 单片机的接口电路及程序扫描和中断扫描的程序设计方法

矩阵键盘的程序设计步骤：

1）检查是否有键按下，其方法是输出扫描码，使所有列线为 0；然后读入行线状态，检查是否有行线为 0。若有，则表明有行线和列线接通，意味着有键按下。

2）去抖动。当有键按下时，延时 10ms 左右，待抖动消失后，在稳定状态下进行被按键识别。

3）按键识别。从第 0 行第 0 列开始，顺序对所有按键编号。通过为 0 的行列坐标确定按键编号。

4）查表得键值。根据扫描得到的键编号查找键盘编码表，获得与被按键功能对应的键码。等待按键释放后，再进行按键功能的处理操作。

（4）行列式键盘与 8255 接口方法及相应的驱动程序设计

13.3 案例设计

13.3.1 硬件电路

行列键盘电路如图 13-3 所示。

图 13-3 行列键盘电路

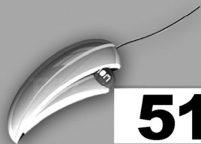
13.3.2 软件代码

（1）汇编程序

```
PORTA       EQU     8FFFH           ;8255A 口地址
PORTB       EQU     9FFFH           ;8255B 口地址
PORTC       EQU     0AFFFH          ;8255C 口地址
CADDR       EQU     0BFFFH          ;8255 控制字地址
SHOW1       EQU     70H             ;数码管 1 位数存放内存位置
SHOW2       EQU     71H             ;数码管 2 位数存放内存位置
SHOW3       EQU     72H             ;数码管 3 位数存放内存位置
SHOW4       EQU     73H             ;数码管 4 位数存放内存位置
SHOW5       EQU     74H             ;数码管 5 位数存放内存位置
SHOW6       EQU     75H             ;数码管 6 位数存放内存位置
SHOW7       EQU     76H             ;数码管 7 位数存放内存位置
SHOW8       EQU     77H             ;数码管 8 位数存放内存位置
KEYNUM      EQU     78H             ;键盘码
KEYCNT      EQU     79H             ;键盘计数器
KEYPRES     BIT     71H             ;按键监视位
            ORG     0000H
            AJMP    INI
            ORG     0100H
INI:        MOV     A,#81H          ;设置 8255 工作方式
            MOV     DPTR,#CADDR
            MOVX    @DPTR,A
            MOV     DPTR,#PORTC
            MOV     A,#00H
            MOVX    @DPTR ,A
            MOV     SHOW1,#00H      ;初始化数码管数组
            MOV     SHOW2,#00H
            MOV     SHOW3,#00H
            MOV     SHOW4,#00H
            MOV     SHOW5,#00H
            MOV     SHOW6,#00H
            MOV     SHOW7,#00H
            MOV     SHOW8,#00H
            CLR     KEYPRES
MAIN:       MOV     KEYNUM,#00H     ;键盘扫描
            ACALL   SCAN
            MOV     A,KEYNUM
            MOV     C,KEYPRES
            JNC     DISP
            MOV     DPTR,#NUMLAB
            MOVC    A,@A+DPTR
            MOV     SHOW1,A
            CLR     KEYPRES
DISP:       ACALL   PLAY
```



```
                JMP         MAIN
/*********************************************/
/*          键盘扫描程序                      */
/* 说明:    使用键盘时，要将跳线跳至右端       */
/*********************************************/
SCAN:           MOV         R0,#04H
                MOV         R1,#80H
SCLOP:          MOV         A,R1
                CPL         A
                MOV         DPTR,#PORTC         ;输出高四位屏蔽码
                MOVX        @DPTR,A
                ANL         A,#0F0H
                MOV         R2,A
                MOVX        A,@DPTR             ;读回扫描结果
                ANL         A,#0FH
                CJNE        A,#0FH,STORE        ;判断是否有按键按下
                JMP         NEXT
STORE:          MOV         KEYNUM,A            ;保存按键位值
                MOV         A,R2
                ORL         A,KEYNUM
                MOV         KEYNUM,A
                ACALL       D1MS
                MOVX        A,@DPTR
                ANL         A,#0FH
                CJNE        A,#0FH,STORE        ;等待按键释放
                MOV         A,KEYCNT
                INC         A
                MOV         KEYCNT,A
                SETB        KEYPRES
                JMP         HANDLE
NEXT:           MOV         A,R1
                RR          A
                MOV         R1,A
                DJNZ        R0,SCLOP
                MOV         KEYNUM,#00H
                JMP         EXIT
HANDLE:         MOV         A,KEYNUM            ;按键位图码转换为按键值
                MOV         KEYNUM,#00H
IF1:            CJNE        A,#0EEH,IF2
                MOV         KEYNUM,#01H
                JMP         EXIT
IF2:            CJNE        A,#0EDH,IF3
                MOV         KEYNUM,#02H
                JMP         EXIT
IF3:            CJNE        A,#0EBH,IF4
                MOV         KEYNUM,#03H
```




```
                JMP         EXIT
IF4:            CJNE        A,#0E7H,IF5
                MOV         KEYNUM,#04H
                JMP         EXIT
IF5:            CJNE        A,#0DEH,IF6
                MOV         KEYNUM,#05H
                JMP         EXIT
IF6:            CJNE        A,#0DDH,IF7
                MOV         KEYNUM,#06H
                JMP         EXIT
IF7:            CJNE        A,#0DBH,IF8
                MOV         KEYNUM,#07H
                JMP         EXIT
IF8:            CJNE        A,#0D7H,IF9
                MOV         KEYNUM,#08H
                JMP         EXIT
IF9:            CJNE        A,#0BEH,IF0
                MOV         KEYNUM,#09H
                JMP         EXIT
IF0:            CJNE        A,#0BDH,IFA
                MOV         KEYNUM,#00H
                JMP         EXIT
IFA:            CJNE        A,#0BBH,IFB
                MOV         KEYNUM,#0AH
                JMP         EXIT
IFB:            CJNE        A,#0B7H,IFC
                MOV         KEYNUM,#0BH
                JMP         EXIT
IFC:            CJNE        A,#7EH,IFD
                MOV         KEYNUM,#0CH
                JMP         EXIT
IFD:            CJNE        A,#7DH,IFE
                MOV         KEYNUM,#0DH
                JMP         EXIT
IFE:            CJNE        A,#7BH,IFF
                MOV         KEYNUM,#0EH
                JMP         EXIT
IFF:            CJNE        A,#77H,EXIT
                MOV         KEYNUM,#0FH
EXIT:           RET
/*********************************************/
/*              显示子程序                     */
/* 说明:           无                          */
/*********************************************/
PLAY:           MOV         R0,#08H
                MOV         R1,#SHOW1          ;取显示码
```

```
            MOV         R2,#80H
DPLOP:      MOV         A,@R1
            MOV         DPTR,#PORTA
            MOVX        @DPTR,A           ;送出个位的 7 段代码
            MOV         DPTR,#PORTB
            MOV         A,R2
            MOVX        @DPTR ,A          ;开相应的位显示
            ACALL       D1MS              ;显示 162μs
            MOV         DPTR,#PORTB
            MOV         A,#0FFH
            MOVX        @DPTR ,A          ;关闭十位显示，防止鬼影
            MOV         A,R2
            RR          A
            MOV         R2,A
            INC         R1
            DJNZ        R0,DPLOP          ;循环执行 8 次
            RET
/********************************************/
/*                  延时程序                  */
/*  说明:       用于数码管显示延时              */
/********************************************/
D1MS:       MOV         R6,#150
            DJNZ        R6,$
            RET
;实验板上的 7 段数码管 0～9 数字的共阴显示代码
NUMLAB:DB   3FH,06H,5BH,4FH,66H,6DH,7DH,07H,7FH,6FH,77H,7CH,39H,5EH,79H,71H
            END
```

（2）C 语言程序

```
#include<reg51.h>
sbit        DQ=P1^2;
sbit        P1_3=0x93;
#define     PORTA       0x8FFF          //8255A 口地址
#define     PORTB       0x9FFF          //8255B 口地址
#define     PORTC       0xAFFF          //8255C 口地址
#define     CADDR       0xBFFF          //8255 控制字地址
#define     uchar       unsigned char
#define     unint       unsigned int
bit         if_keypress;
uchar       keypresscnt;
uchar       Keynum;
void        WriteData(unsigned int Addr,unsigned char Data);
uchar       ReadData(unsigned int Addr);
void        Printer(void);
void        Delay(void);
```

```
void            Ini(void);
uchar           Keyboard(void);
uchar           Handlekey(void);
unint           show[8];
code uchar   word[16]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,
0x7c,0x39,0x5e,0x79,0x71};
void main(void)
{
        Ini();
        while(1)
        {
                Keynum=Handlekey();        //按键处理，读取按键值的操作在线程中只能出现一次，否
则会因为多次扫描出现滞后或其他错误
                 if(if_keypress)
                       show[7]=word[Keynum];
                 if_keypress=0;
                Printer();
        }
}
/*********************************************/
/*                  初始化程序                     */
/*  说明:     配置寄存器及初始化参数               */
/*********************************************/
void Ini(void)
{
        if_keypress=0;
        keypresscnt=0;
        Keynum=0;
                                                        //8255 工作模式
        WriteData(CADDR,0x81);
        WriteData(PORTC,0xff);
}

/*********************************************/
/*                  写外部寄存器                   */
/*  说明:              无                          */
/*********************************************/
void WriteData(unint Addr,uchar Data)
{
       *((uchar xdata *)Addr)=Data;
}
/*********************************************/
/*                  读外部寄存器                   */
/*  说明:              无                          */
/*********************************************/
```

```
unsigned char ReadData(unsigned int Addr)
{
    return *((unsigned char xdata *)Addr);
}
/*******************************************/
/*              延时程序                   */
/* 说明:          无                       */
/*******************************************/
void Delay(void)
{
     int i,j;
     for(i=0;i<20;i++)
          for(j=0;j<5;j++);
}
/*******************************************/
/*            键盘扫描程序                 */
/* 说明:                                   */
/*******************************************/
unsigned char Keyboard(void)
{
    unsigned  char  scan,keypress=0x00;              //初始化，否则当有按键按下时，程序将
默认为初值
     unsigned int    i;
     unsigned char MASK,mask=0x10;
     for(i=0;i<4;i++)
     {
          MASK=~(mask);
          mask=mask<<1;

          WriteData(PORTC,MASK);
          scan=ReadData(PORTC);
          if((scan&0x0f)!=0x0f)
          {
               if_keypress=1;
               keypresscnt++;
               keypress=scan;
          }
          while((ReadData(PORTC)&0x0f)!=0x0f);
     }
      return keypress;
}
/*******************************************/
/*            键盘读取程序                 */
/* 说明:    返回按下按键的值               */
/*******************************************/
```

```
uchar Handlekey(void)
{
        unsigned char handlekey;
        switch(Keyboard())
        {
                case 0xee:
                        handlekey=0x01;
                        break;
                case 0xed:
                        handlekey=0x02;
                        break;
                case 0xeb:
                        handlekey=0x03;
                        break;
                case 0xe7:
                        handlekey=0x04;
                        break;
                case 0xde:
                        handlekey=0x05;
                        break;
                case 0xdd:
                        handlekey=0x06;
                        break;
                case 0xdb:
                        handlekey=0x07;
                        break;
                case 0xd7:
                        handlekey=0x08;
                        break;
                case 0xbe:
                        handlekey=0x09;
                        break;
                case 0xbd:
                        handlekey=0x00;
                        break;
                case 0xbb:
                        handlekey=0x0a;
                        break;
                case 0xb7:
                        handlekey=0x0b;
                        break;
                case 0x7e:
                        handlekey=0x0c;
                        break;
                case 0x7d:
```

```
                handlekey=0x0d;
                break;
            case 0x7b:
                handlekey=0x0e;
                break;
            case 0x77:
                handlekey=0x0f;
                break;
            default:
                handlekey=0xff;
                break;
        }
        return handlekey;
}
/*********************************************/
/*              显示子程序                     */
/* 说明:              无                       */
/*********************************************/
void Printer(void)
{
        unint i;
        uchar MASK,mask=0x01;
        for(i=0;i<8;i++)
        {
            MASK=mask;
            mask=mask<<1;

            WriteData(PORTA,show[i]);
            WriteData(PORTB,MASK);
            Delay();
            WriteData(PORTA,0x00);
            WriteData(PORTB,MASK);
        }
}
```

13.4 案例笔记：C51 精炼的判键程序

以下的按键程序非常精炼，但是用到了许多初学者不常使用的 C 语言知识。

```
typedef struct {
        unsigned char Name;
        void (*Func_CLOSE)(void);
        void (*Func_OPEN)(void);
        }BUTTON;
```

```
#define BUTTONmax 8
const BUTTON FunTab[BUTTONmax]={
        ~0x01,PA0_CLOSE,PA0_OPEN,
        ~0x02,PA1_CLOSE,PA1_OPEN,
        ~0x04,PA2_CLOSE,PA2_OPEN,
        ~0x08,PA3_CLOSE,PA3_OPEN,
        ~0x10,PA4_CLOSE,PA4_OPEN,
        ~0x20,PA5_CLOSE,PA5_OPEN,
        ~0x40,PA6_CLOSE,PA6_OPEN,
        ~0x80,PA7_CLOSE,PA7_OPEN,
        //可扩展组合键
};

#define AllButtonOPEN 0xFF
unsigned char InputTimes,InputData=AllButtonOPEN,InputNew;
const BUTTON *pButton=FunTab;
void ScanInput(void)
{

  if (InputData == PINA)
        {
       if (InputNew)
         {
         InputTimes++;
         if (InputTimes == 8) //判键的防抖动
              {//确认一次按键(包括按下和松开)
              InputTimes = 0;
              InputNew = 0;
              if (InputData == AllButtonOPEN) pButton->Func_OPEN();
              else
                   {
                for (pButton = FunTab;pButton < FunTab+BUTTONmax;pButton++)
                         {
                         if (pButton->Name == InputData)
                              {
                              pButton->Func_CLOSE();
                              break;
                              }
                         }
                   }
              }
```

```
        }
    }
else
    {
    InputTimes = 0;
    InputData = PINA;
    InputNew = 1;
    }
```





案例14



中 断

14.1 案例任务

利用边沿中断触发、低电平中断触发、查询方式三种方法使连接 P32 的按键按下一次，P1.4 所接发光二极管亮灭状态改变。

14.2 案例要点

（1）独立式按键工作原理

（2）基本概念

1）中断：在微处理器执行现行程序的过程中，发生随机事件或特殊请求时，即中止现行程序，待处理完毕后，再返回被中止的程序继续执行，这个过程称为中断，中断是硬件或软件激发的一次调用，中止当前执行的程序而调用一个过程。实现中断的硬件逻辑和实现中断功能的指令，通称为中断系统。引起中断的事件称为中断源。实现中断功能的处理程序称为中断服务程序。

2）中断请求：中断源向微处理器发出的请求，每个中断源都能发出中断请求信号。请求是随机的，中断请求一旦发生就应保持这一请求直至微处理器响应该中断，请求信号才被清除。

3）中断响应：微处理器对外部中断请求信号的回答。具体地说就是暂停现行程序，转去执行为该请求服务的中断服务程序。

4）优先权：为每个中断源确定中断优先级别。

5）中断嵌套：当微处理器正在处理一个中断源请求的时候，又发生了另一个优先级比当前中断源高的中断请求，微处理器暂时终止执行当前中断服务程序，转而去处理优先级更高的中断请求，处理完毕后，再继续执行当前的程序。

6）可屏蔽中断：受中断允许程序控制影响的中断源。

7）不可屏蔽中断：不受中断允许程序控制影响的中断源。

8）中断向量：中断服务程序的入口地址。

9）中断向量地址：保存中断向量的地址。

10）能提出中断请求的计算机内部电路与外设称为中断源。51 单片机中断源漫画如图 14-1 所示。

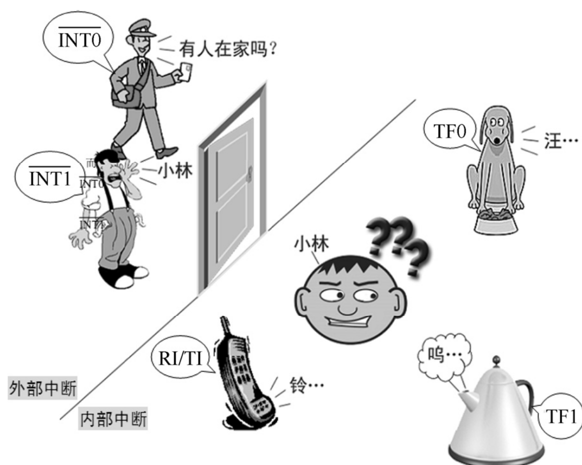


图 14-1 51 单片机的中断源

（3）中断的作用

1）中断技术能实现 CPU 与多个外设并行工作，提高了 CPU 的利用率及数据的输入/输出效率。

2）中断技术能对单片机运行过程中某个出现的事件或突然发生的故障做到及时发现并及时处理，即实现实时处理。

3）可实现多道程序的切换运行。

4）实现实时控制。

5）在多机系统中，实现各处理机之间的信息交换和任务切换。

（4）51 单片机的中断系统结构

1）51 系列的中断系统简单实用，其基本特点：有 5 个固定的可屏蔽中断源，3 个在片内，2 个在片外，它们在程序存储器中各有固定的中断入口地址，由此进入中断服务程序；5 个中断源有两级中断优先级，可形成中断嵌套；2 个特殊功能寄存器用于中断控制和条件设置的编程。

2）5 个中断源的符号、名称及产生的条件如下：

INT0：外部中断 0，由 P3.2 端口线引入，低电平或下跳沿引起。

INT1：外部中断 1，由 P3.3 端口线引入，低电平或下跳沿引起。

T0：定时器/计数器 0 中断，由 T0 计满回零引起。

TI：定时器/计数器 1 中断，由 T1 计满回零引起。

TI/RI：串行 I/O 中断，串行端口完成一帧字符发送/接收后引起。

3）外部中断有下跳沿引起和低电平引起的选择；串行中断有发送（TI）和接收（RI）的区别；各个中断源打开与否，受中断自身的允许位和全局允许位的控制，并具有高优先级和低优先级的选择。

（5）中断控制寄存器

中断系统有两个控制寄存器 IE 和 IP，它们分别用来设定各个中断源的打开/关闭和中断优先级。此外，在 TCON 中另有 4 位用于选择引起外部中断的条件并作为标志位。

（6）理解中断请求标志的产生和作用



（7）中断的响应过程

1）保护断点：保存下一条将要执行指令的地址，方法是把其地址送入堆栈。

2）寻找中断入口：由单片机硬件自动寻找 5 个不同中断源的入口地址。

3）执行中断处理程序。

4）中断返回：执行完中断指令后，就从中断处返回到主程序，继续执行相关主程序指令。

正常的情况下，从中断请求信号有效开始，到中断得到响应，通常需要 3～8 个机器周期。中断得到响应后，自动清除中断请求标志（对串行 I/O 端口的中断标志，要用软件清除），将断点即程序计数器之值（PC）压入堆栈（以备恢复用）；然后把相应的中断入口地址装入 PC，使程序转入到相应的中断服务程序中执行。

（8）51 单片机的中断优先级

中断源的同级内部自然优先级（由低到高）顺序为外部中断 0、定时器 T0、外部中断 1、定时器 T1、串行口。

（9）51 单片机的中断向量地址

5 个独立中断源中断向量地址如图 14-2 所示。

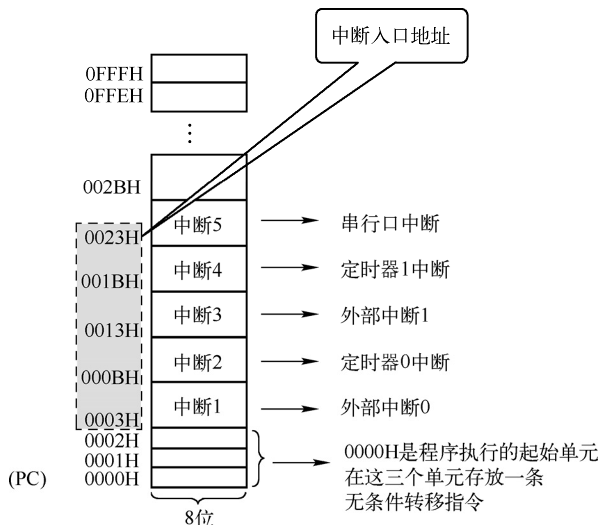


图 14-2　中断向量地址

由于各个中断入口地址相隔甚近，不便于存放较长的中断服务程序，故通常在中断入口地址开始的二三个单元中，安排一条转移类指令，以转入其他地址的中断服务程序。

14.3　案例设计

14.3.1　硬件电路

中断电路如图 14-3 所示。

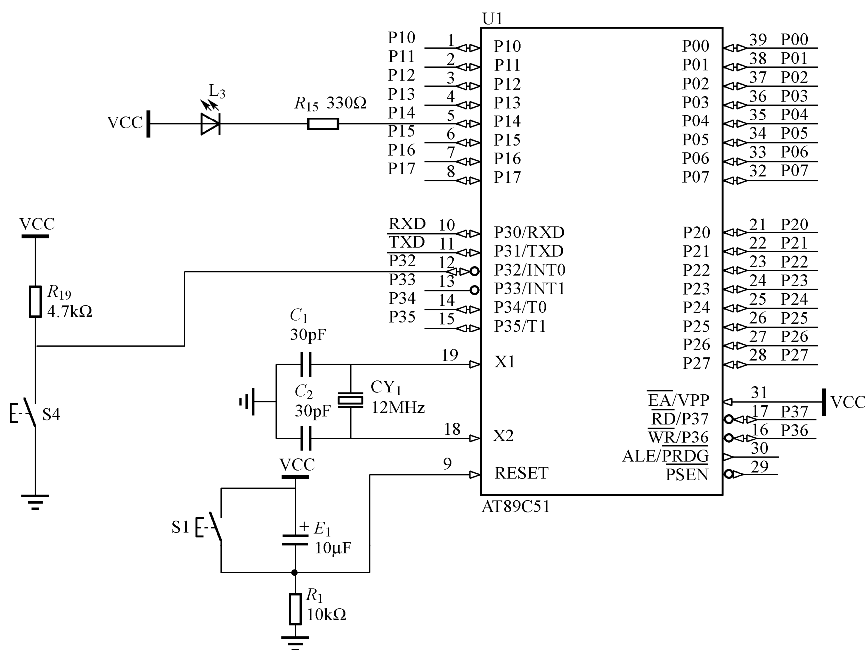


图 14-3 中断电路

▷▷▷ 14.3.2 软件代码

1．低电平触发中断

（1）汇编程序

```
          ORG     0000H
          AJMP    MAIN
          ORG     0003H
          AJMP    PINT0          ;跳转到中断服务程序
          ORG     0100H
MAIN:     MOV     SP,#40H
          SETB    EA             ;开总允许开关
          SETB    EX0            ;开 INT0 中断
          CLR     IT0            ;低电平触发中断
H:        SJMP    H
          ORG     0200H
PINT0:    CPL     P1.4           ;中断服务程序
WAIT:     JNB     P3.2,WAIT      ;等待按键释放
          RETI
          END
```

（2）C 语言程序

```
#include <AT89X51.h>        //包括一个 51 标准内核的头文件
sbit s4=P3^2;
unsigned int i;
```

```
void main(void)              //主程序
{
    IT0=0;                   //外中断电平触发
    EA=1;                    //打开总中断
    EX0=1;
    while(1)                 //主程序循环
    {
    }
}
int0() interrupt 0           //外中断 0
{
    P1_4=~P1_4;              //在中断里点亮 LED
    while(~s4)
    {
    }
}
```

2．负跳变触发中断

（1）汇编程序

```
            ORG     0000H
            SJMP    MAIN
            ORG     0003
            AJMP    PINT0
            ORG     0056H
MAIN:       SETB    EA          ;允许总中断
            SETB    EX0         ;开 INT0 中断
            SETB    IT0         ;负跳变触发
            SJMP    $
PINT0:      CPL     P1.4        ;取反
            RETI
            END
```

（2）C 语言程序

```
#include <AT89X51.h>         //包括一个 51 标准内核的头文件
sbit s4=P3^2;
unsigned int i;
void main(void)              //主程序
{
    IT0=1;                   //外中断边沿触发，产生中断
    EA=1;                    //打开总中断
    EX0=1;
    while(1)                 //主程序循环
    {
    }
}
```

```
int0() interrupt 0           //外中断 0
{
        P1_4=~P1_4;          //在中断里点亮 LED
}
```



3．查询方式

（1）汇编程序

```
S4          BIT      P3.2
            ORG      0000H
            AJMP     MAIN
            ORG      0100H
MAIN:       JB       S4,MAIN
WAIT:       JNB      S4,WAIT
            LCALL    DELY10MS        ;消抖
            JNB      S4,WAIT
            CPL      P1.4
            JMP      MAIN
DELY10MS:   MOV      R6,#20
D1:         MOV      R7,#248
            DJNZ     R7,$
            DJNZ     R6,D1
            RET
            END
```

（2）C 语言程序

```
#include<AT89X51.h>
sbit s4=P3^2;
void main (void)
{
     unsigned int i;
     EA=1;
     EX0=1;
     IT0=1;
     while(1)
     {
          if(~s4)
          {
               for(i=0;i<1000;i++)     //消抖
               if(~s4)
               {
                P1_4=~P1_4;
               }
          }
     }
}
```



14.4 案例笔记

14.4.1 利用 51 单片机的中断系统实现三级以上中断嵌套

51 单片机在设计上由于受功能和部件封装等因素的限制，在中断系统的管理上较为简单。其采用内部固定矢量法实现，给用户只留有 5 个中断源，且中断优先级仅有两级。然而，利用软件设计的灵活性，只需要增加几条指令，就可实现三级以上的中断嵌套，提高了单片机的中断处理能力。

在中断系统中，中断优先级的控制主要决定于两个部件，其一是中断优先级寄存器（IP），通过软件编程控制；其二是优先级触发器，此触发器是不可寻址的。要实现单片机多级中断嵌套，关键在于如何清除低优先级状态触发器的中断标志，才能使 CPU 响应中断请求。在指令集中，只有 RETI 指令可清除优先级触发器，但 CPU 若执行 RETI 后，程序就会退出中断服务。为了既能消除优先级触发器的标志，又能使程序不退出中断，可在中断服务程序开始补一条 RETI 指令，这是一条非正常中断返回指令。在此指令之前，将这一指令之后的第一条指令地址压入堆栈，中断服务程序执行完该 RETI 指令后，清除了中断优先级触发器的状态，同时被弹回到该 RETI 后面的第一条指令的地址处，使程序继续执行中断服务，为响应高一级中断做好了准备。

如某系统设有三级中断，分别是 T0（内部定时器 0）最高，INT0（外部中断 0）其次，INT1（外部中断 1）最低。在初始化程序中设置片内中断优先级控制器 IP 为 PT0 位置“1”，PX0、 PX1 位置“0”。即 T0 为高优先级，INT0、INT1 为低优先级。若不考虑 INT0 和 INT1 之间的中断嵌套，根据 51 单片机内部中断源的优先级排队，在同一级上，INT0 优先于 INT1。若进入 INT 1 的中断后，INT0 无法中断它。所以，在 INT0 和 INT1 之间用软件实现中断优先级的排队。在 INT1 中断服务开始处，利用 RETI 指令清除中断优先级触发器的状态，使 CPU 能响应 INT0 的中断请求，而 INT0 为正常的中断服务程序，它可利用优先级触发器屏蔽 INT1 的中断，使 INT0 的中断优先级高于 INT1。

其中断服务程序片断如下：

```
CTCO:                                   ; T0 中断服务程序
            RETI
INT0:                                   ;INT0 中断服务程序
            RETI
INT1:       PUSH     DPH                ;INT1 中断服务程序
            PUSH     DPL
            MOV      DPTR,#JXDZ
            PUSH     DPL
            PUSH     DPH
            RETI
JXDZ:       NOP
            ⋮
            POP      DPL
```

```
        POP     DPH
        RETI
```

通常 CPU 的中断响应时间为 3～8 个机器周期，如果 CPU 正在处理同级或更高级的中断，额外的等待时间将取决于正在执行的中断服务程序的处理过程。由于 INT0 的中断优先级是通过软件编程设置的，INT0 的中断响应时间会稍长一些。在 INT1 的中断服务程序中，从标号 INT0 执行到标号 JXDZ 共需要 12 个机器周期，因此，INT0 中断响应时间最长为 15～20 个机器周期。

这样，实现了在 INT1 的中断服务中嵌套 INT0 的中断服务。同时，由于 T0 的中断为高一级的中断优先级，所以在 INT0 或 INT1 的中断服务中，又允许嵌套 T0 的中断服务，从而实现了 T0<INT0<INT1 的三级中断嵌套。

14.4.2 中断与调用子程序的区别

子程序的执行是由程序员事先安排好的（由一条调用子程序指令来转入），而中断服务程序是由随机的中断事件引起的。子程序的执行受到主程序或上一级子程序的控制，而中断服务程序一般与被中断的程序毫无关系。有可能发生多个中断事件同时请求 CPU 服务的情况。中断服务程序和子程序区别见表 14-1。

表 14-1 中断服务程序和子程序的区别

		子 程 序	中断服务程序
同		要保护什么内容，才能保证返回断点后正常工作	
异	调用特点	子程序在确定的指令处调用	对强迫中断的服务程序具有随机性
	处理内容	程序中的重复内容	中断一般处理 I/O 操作
	返回指令	RET	RETI

案例15

 计　数　器

▷▷ 15.1　案例任务

当按下 S5 键，数码管显示加 1，从 00～255，加到 256 返回 00。推荐用定时/计数器的计数完成该功能。

▷▷ 15.2　案例要点

（1）了解定时器/计数器 T0 和 T1 的内部结构、工作原理及功能

T0 和 T1 有两种功能：定时和计数，如图 15-1 所示，计数就是计算事件的发生次数。霍尔开关每当车轮转过一圈时磁铁接近一次霍尔开关，于是就会输出一个脉冲。如果把这个脉冲输入单片机，单片机可以在每次脉冲到来时计一个数，假设 60s 内单片机的计数值为 n，车轮每转过一圈的时间为 T，则定时时间 60 和计数 n 的关系为

$$60 = nT$$

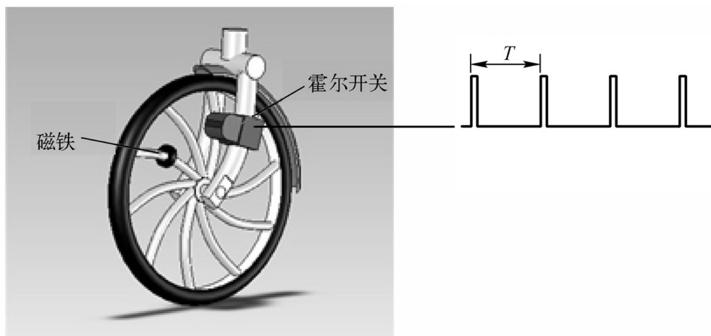


图 15-1　计数和定时

定时器和计数器的区别和联系见表 15-1。

表 15-1　定时器和计数器的区别

	定　时　器	计　数　器
时钟源	内部时钟脉冲	外部输入引脚 T0 或 T1
计数方式	每个机器周期产生一个脉冲使计数器增 1	当 T0（P3.4）和 T1（P3.5）产生负跳变时，计数器值增 1

1）计数功能。

启动后，对外部输入脉冲（负跳变）进行加 1 计数，T0 的脉冲由 P3.4 输入，T1 的脉冲由 P3.5 输入。

计数器加满溢出时，将中断标志位 TF0/TF1 置 1，向 CPU 申请中断。

计数脉冲个数=溢出值-计数初值

上述计数公式如图 15-2 所示。

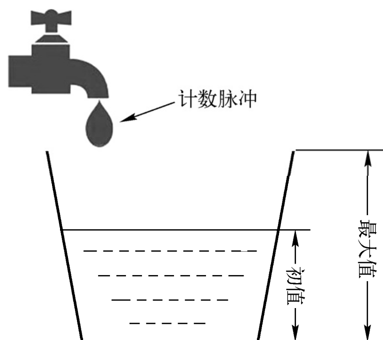


图 15-2 计数脉冲个数和初值、最大值的关系

计数实例：对零件和产品的计数、对大桥和高速公路上车流量的统计等。

2）定时功能。

启动后，开始定时，定时时间到，中断标志位 TF0/TF1 自动置 1，向 CPU 申请中断。

定时功能也是以计数方式来工作的，此时是对单片机内部的脉冲进行加 1 计数，此脉冲的周期正好等于机器周期。

定时时间=（溢出值-计数初值）×机器周期

定时实例：一天 24 小时的计时（称为日时钟）；在监测系统中，对被测点的定时取样；在读键盘时，为了去抖，一般延迟一段时间再读信号。

（2）特殊功能寄存器 TMOD（定时器控制寄存器）和 TCON（工作模式寄存器）的程序设置方法

（3）了解定时/计数器的四种工作方式，以及各自的应用场合

T0 有四种工作方式，T1 有三种工作方式。

1）方式 0：13 位定时/计数器方式；溢出值是 2^{13}=8192。

2）方式 1：16 位定时/计数器方式；溢出值是 2^{16}=65536。

3）方式 2：8 位自动重装初值定时/计数器方式；溢出值是 2^{8}=256。

4）方式 3：T0 分成两个独立的 8 位计数器方式。

上述方式中重点掌握方式 1、方式 2 的应用。因为方式 0 的最大计数值为 $2^{13}-1$，计数能力不及方式 1（最大计数值为 $2^{16}-1$），所以可用方式 1 代替方式 0，方式 0 的存在是为了兼容前一代的 51 单片机；方式 3 将 T0 分成两个独立的 8 位计数器 TL0 和 TH0，这两个 8 位计数器功能并不完整，方式 3 的存在是为了考虑当时硬件资源紧张，而现在有多个计数器的 51 系列单片机，因此方式 3 也不常用；方式 2 由于采用硬件重装的方式，因此可用于精确定时。

（4）计数满/定时到时的处理

（5）掌握定时/计数器初值的计算方法

（6）掌握定时/计数器的初始化及应用程序设计

▷▷ 15.3 案例设计

▷▷▷ 15.3.1 硬件电路

计数器电路如图 15-3 所示。

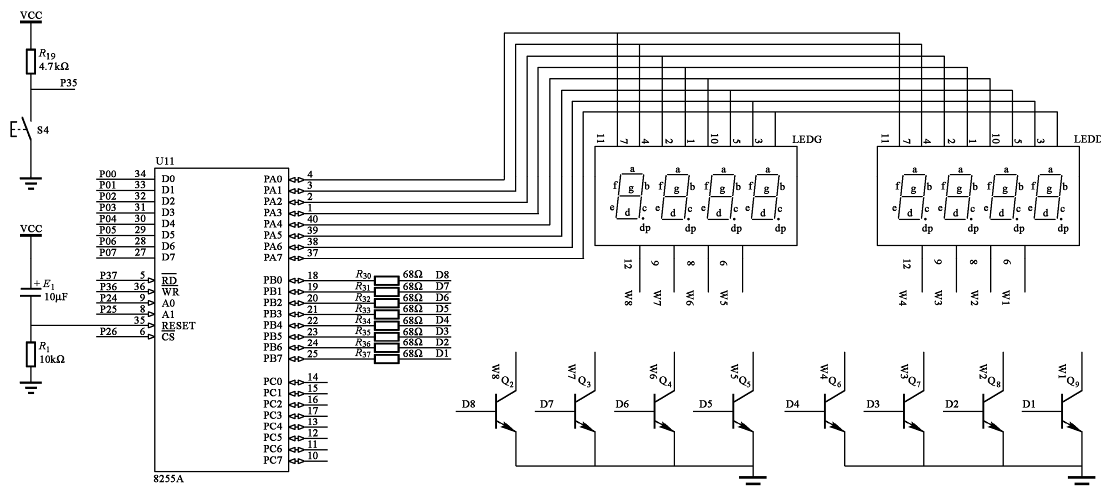


图 15-3 计数器电路

▷▷▷ 15.3.2 软件代码

（1）汇编程序

```
PORTA     EQU     8FFFH          ;8255A 口地址
PORTB     EQU     9FFFH          ;8255B 口地址
PORTC     EQU     0AFFFH         ;8255C 口地址
CADDR     EQU     0BFFFH         ;8255 控制字地址
          ORG     0000H
          SJMP    MAIN
          ORG     0030H
MAIN:     MOV     A,#80H         ;方式 0
          MOV     DPTR,#CADDR
          MOVX    @DPTR,A        ;设置 8255 工作方式
          SETB    P2.6
          MOV     TMOD,#060H     ;T1，与 INT0 无关，计数，方式 0
          MOV     TL1,#00H       ;初值
          MOV     TH1,#00H
```



```
            SETB    TR1
LOOP:       MOV     A,TL1
            MOV     B,#100
            DIV     AB
            MOV     R2,A            ;R2 为百位
            MOV     A,B
            MOV     B,#10
            DIV     AB
            MOV     R1,A            ;R1 为十位
            MOV     R0,B            ;R0 为个位
            MOV     A,#80H
            MOV     DPTR,#PORTB
            MOVX    @DPTR,A         ;置个位数码管有效
            MOV     DPTR,#ZXM
            MOV     A,R0
            MOVC    A,@A+DPTR       ;设置个位显示码
            MOV     DPTR,#PORTA
            MOVX    @DPTR,A
            CALL    DELAY
            MOV     DPTR,#ZXM
            MOV     A,R1
            MOVC    A,@A+DPTR       ;设置十位显示码
            MOV     DPTR,#PORTA
            MOVX    @DPTR,A
            MOV     A,#40H          ;置十位数码管有效
            MOV     DPTR,#PORTB
            MOVX    @DPTR,A
            CALL    DELAY
            MOV     DPTR,#ZXM
            MOV     A,R2
            MOVC    A,@A+DPTR       ;设置百位显示码
            MOV     DPTR,#PORTA
            MOVX    @DPTR,A
            MOV     A,#20H          ;置百位数码管有效
            MOV     DPTR,#PORTB
            MOVX    @DPTR,A
            CALL    DELAY
            LJMP    LOOP
DELAY:      MOV     R6,#10          ;延时子程序
LOOP1:      MOV     R7,#10
            DJNZ    R7,$
            DJNZ    R6,LOOP1
                    RET
ZXM:        DB  3fh,06h,5bh,4fh,66h,6dh,7dh,07h,7fh,6fh
                    END
```

（2）C 语言程序

```
#include<reg51.h>
#include<absacc.h>
#include<intrins.h>
#define uchar unsigned char
#define uint unsigned int
//8255 C 语言接口程序
#define PORTA   &XBYTE[0x8FFF]          //8255A 口地址
#define PORTB   &XBYTE[0x9FFF]          //8255B 口地址
#define PORTC   &XBYTE[0x0AFFF]         //8255C 口地址
#define CADDR  &XBYTE[0x0BFFF]          //8255 控制字地址
uchar showdata[8]={1,2,3,4,5,6,7,8};
//*************数码管编码***共阴 0-F-无-负号*********************************
uchar showcode[10]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};
#define DelayTime 100
void write_8255(uchar xdata * ad_adr,uchar dat)
{
        *ad_adr=dat;
}
void init_8255(void)
{
        write_8255(CADDR,0x81);  //设置 A 口为工作方式 0，输出；C 口低四位设置为输入
                                 //设置 B 口为工作方式 0，输出；C 口高四位设置为输出

        write_8255(PORTA,0x00);          //设置 A 口为 0x00
        write_8255(PORTB,0x00);          //设置 B 口为 0x00
        write_8255(PORTC,0xff);          //设置 C 口为 0xff
}
void Init_Time1(void)
{
        TMOD=0x60;                       //设置定时器 1 为工作方式 1，16 位定时器
        TH1=0x00;
        TL1=150;
        TR1=1;
}
/*************精确延时函数*************************************/
//延时(14+6*i)µs，i 最小为 0，最大为 255，即 1544µs
void Delayus(uchar i)
{
        while(i--);
}
void main(void)
{uchar Time,temp;
        init_8255();
```

```
        Init_Time1();
        while(1)
        {
            Time=150;//TL1;
                showdata[2]=Time/100;
                temp=Time-showdata[2]*100;
                showdata[1]=temp/10;
                showdata[0]=temp%10;
            write_8255(PORTB,0x00);                         //设置 B 口
            write_8255(PORTA,showcode[showdata[0]]);        //设置 A 口
            write_8255(PORTB,0x80);                         //设置 B 口
            Delayus(DelayTime);
                                                            /数码管 2
            write_8255(PORTB,0x00);                         //设置 B 口
            write_8255(PORTA,showcode[showdata[1]]);        //设置 A 口
            write_8255(PORTB,0x40);                         //设置 B 口
            Delayus(DelayTime);
                                                            //数码管 2
            write_8255(PORTB,0x00);                         //设置 B 口
            write_8255(PORTA,showcode[showdata[2]]);        //设置 A 口
            write_8255(PORTB,0x20);                         //设置 B 口
            Delayus(DelayTime);
            write_8255(PORTB,0x00);                         //设置 B 口
        }
    }
```

▷▷ 15.4 案例笔记：定时/计数器四种工作方式总结

定时/计数器四种工作方式总结见表 15-2。

表 15-2 定时/计数器四种工作方式总结

	方式 0	方式 1	方式 2	方式 3
M1M0	00	01	10	11
适用计数器	T0、T1	T0、T1	T0、T1	T0
计数最大值	8192	65536	256	256
定时最大值（12MHz）/ms	8.192	65.536	0.256	0.256
是否重设初值	是	是	否	是
用途	不用	常用	T1 的方式 2 常用于波特率发生器	基本不用

四种工作方式中，方式 0、1、3 每次溢出都归 0，如图 15-4 所示，所以需要软件重设初值；而方式 2 溢出后初值保留，如图 15-5 所示，因此称它为硬件重设初值。这两类设初值的方式和软件、硬件实现同一功能的特点类似，即硬件设初值会节省时间，但占用硬件资源，因此最大计数值比方式 0、1 都小。

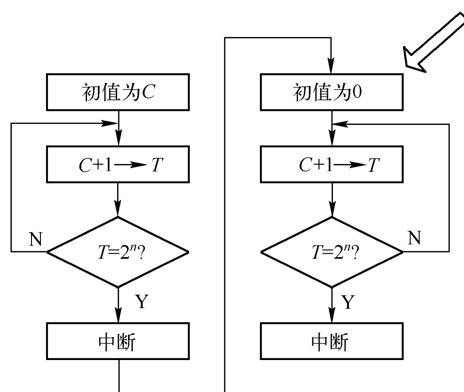


图 15-4　方式 0、1、3 的计数过程

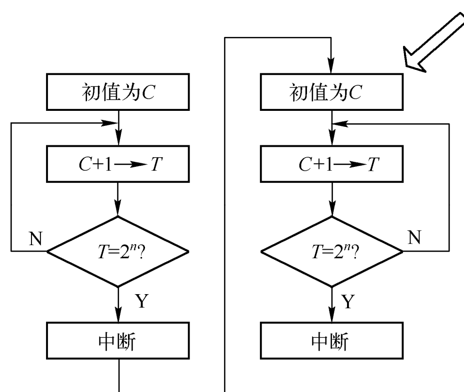


图 15-5　方式 2 的计数过程

案例16



定时秒表



16.1 案例任务

1）定时器方式 1 定时 50ms，20 次（1s）后 LED 显示，直到 59，再加 1 归至 01 循环。

2）为了与用硬件延时相比较，编写纯软件延时 1s 后增 1 显示的程序。

16.2 案例要点

1）长定时的解决办法：硬件定时+硬件计数、硬件定时+软件计数、软件定时。

2）其他要点与案例 15 要点相同。

16.3 案例设计

16.3.1 硬件电路

定时秒表显示电路如图 16-1 所示。

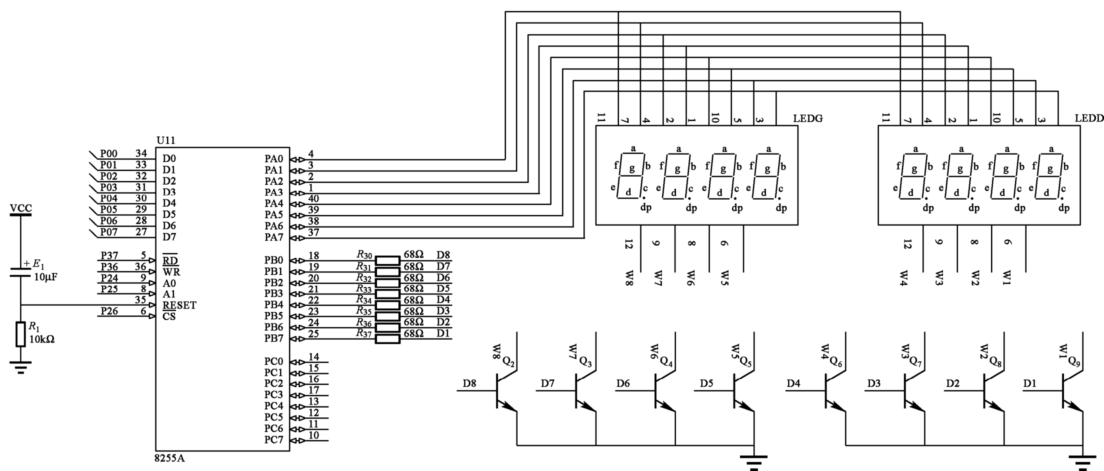


图 16-1 定时秒表显示电路

▷▷▷ 16.3.2 软件代码

1．硬件定时器法延时

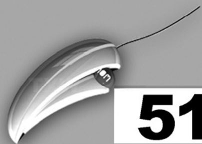
（1）汇编程序

```
ADDR_A   EQU     08FFFH                  ;8255A 口地址为 8FFFH
ADDR_B   EQU     09FFFH                  ;8255B 口地址为 9FFFH
ADDR_C   EQU     0AFFFH                  ;8255C 口地址为 0AFFFH
ADDR_K   EQU     0BFFFH                  ;8255 控制字地址为 0BFFFH
         ORG     0000H
         AJMP    MAIN
         ORG     00BH
         AJMP    INTER
         ORG     0030H
MAIN:    MOV     DPTR,#ADDR_K
         MOV     A,#80H
         MOVX    @DPTR,A
         MOV     DPTR,#ADDR_C
         MOV     A,#0FFH
         MOV     TMOD,#01H          ;T0，与 INT0 无关，定时器，方式 1
         MOV     TH0,#03CH
         MOV     TL0,#0B0H          ;设置初值定时 50ms
         MOV     R0,#00H
         MOV     R1,#00H
         MOV     R2,#00H
         MOV     R3,#20
         SETB    EA
         SETB    ET0
         SETB    TR0
DISPLAY: MOVX    @DPTR,A
         MOV     DPTR,#ADDR_B
         MOV     A,#80H             ;选中个位 LED
         MOVX    @DPTR,A
         MOV     A,R0               ;送个位数据
         MOV     DPTR,#ZXM
         MOVC    A,@A+DPTR
         MOV     DPTR,#ADDR_A
         MOVX    @DPTR,A
         ACALL   DELAY
         MOV     DPTR,#ADDR_B
         MOV     A,#40H             ;选中十位 LED
         MOVX    @DPTR,A
         MOV     A,R1               ;送十位数据
         MOV     DPTR,#ZXM
         MOVC    A,@A+DPTR
         MOV     DPTR,#ADDR_A
```

```
        MOVX    @DPTR,A
        ACALL   DELAY
        SJMP    DISPLAY
DELAY:  MOV     R5,#02H
L1:     MOV     R6,#0E0H
L2:     DJNZ    R6,L2
        DJNZ    R5,L1
        RET
INTER:  PUSH    PSW
        PUSH    ACC
        MOV     TH0,#0DCH           ;重新设初值
        MOV     TL0,#00H
        DJNZ    R3,RTRN             ;50ms*20=1s
        MOV     R3,#20
        INC     R2
        CJNE    R2,#60,GOON         ;是否到 60s
        MOV     R2,#00H
GOON:   MOV     A,R2
        MOV     B,#10
        DIV     AB
        MOV     R1,A                ;保存十位
        MOV     R0,B                ;保存个位
RTRN:   POP     ACC
        POP     PSW
                RETI
ZXM:    DB      3fh,06h,5bh,4fh,66h,6dh,7dh,07h,7fh,6fh,40H
                END
```

（2）C 语言程序

```
#include<reg51.h>
#include<absacc.h>
#include<intrins.h>
#define uchar unsigned char
#define uint unsigned int
uchar showdata[8]={1,2,3,4,5,6,7,8};
//*************数码管编码***共阴 0-F-无-负号********************************
uchar showcode[18]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,
                    0x7C,0x39,0x5E,0x79,0x71,0x00,0x40};
//8255 C 语言接口程序
#define PORTA  &XBYTE[0x8FFF]          //8255A 口地址
#define PORTB  &XBYTE[0x9FFF]          //8255B 口地址
#define PORTC  &XBYTE[0x0AFFF]         //8255C 口地址
#define CADDR  &XBYTE[0x0BFFF]         //8255 控制字地址
//定时器 0 的初始化函数
uchar RealTime_555=0;
uchar Time=0;
```



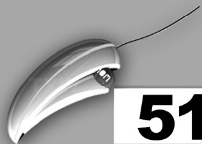

```
void Init_Time0(void)
{
        TMOD = 0x01;                            //设置定时器 0 为工作方式 1，16 位定时器
        TH0=0x3C;
        TL0=0XB0;
        ET0=1;                                  //打开 T0 中断
       EA=1;
        TR0=1;
}
//定时器 0 的中断函数
void Time0() interrupt 1 using 1                //T0 中断函数
{
        TH0=0x3C;
        TL0=0XB0;
        RealTime_555++;
        if(RealTime_555>=20)
        {
                RealTime_555=0;
                Time++;
                if(Time>=60)
                        Time=0;
        }
}
void write_8255(uchar xdata * ad_adr,uchar dat)
{
        *ad_adr=dat;
}
void init_8255(void)
{
        write_8255(CADDR,0x81);         //设置 A 口为工作方式 0，输出；C 口低四位设置为输入
                                        //设置 B 口为工作方式 0，输出；C 口高四位设置为输出
        write_8255(PORTA,0x00);         //设置 A 口为 0x00
        write_8255(PORTB,0x00);         //设置 B 口为 0x00
        write_8255(PORTC,0xff);         //设置 C 口为 0xff
}
void main(void)
{
        init_8255();
        Init_Time0();
        while（1）
        {
                showdata[1]=Time/10;
                showdata[0]=Time%10;
                write_8255(PORTB,0x00);                         //设置 B 口
                write_8255(PORTA,showcode[showdata[0]]);        //设置 A 口
                write_8255(PORTB,0x80);                         //设置 B 口
```

```
                                                          //数码管 2
        write_8255(PORTB,0x00);                           //设置 B 口
        write_8255(PORTA,showcode[showdata[1]]);          //设置 A 口
        write_8255(PORTB,0x40);                           //设置 B 口
        write_8255(PORTB,0x00);                           //设置 B 口
    }
}
```

2．纯软件延时

（1）汇编程序

```
VADIGIT     EQU     20H             ;数码管个位数存放内存位置
VBDIGIT     EQU     21H             ;数码管十位数存放内存位置
COUNT       EQU     30H             ;增 1 计数器
            ORG     0000H
            JMP     MAIN
            ORG     0100H
MAIN:       MOV     DPTR,#0BFFFH
            MOV     A,#80H
            MOVX    @DPTR,A
            MOV     DPTR,#8FFFH
            MOV     A,#0FFH
            MOVX    @DPTR,A
            MOV     DPTR,#0AFFFH
            MOV     A,#0FFH
            MOVX    @DPTR,A
LOOP:       MOV     COUNT,#00H          ;1s 改变一次显示数
LOOP1:      MOV     A,COUNT
            MOV     B,#10
            DIV     AB
            MOV     VBDIGIT,A           ;十位在 A
            MOV     VADIGIT,B           ;个位在 B
            MOV     r0,#50              ;20ms 重新送一次显示数
LOOP2:      MOV     DPTR,#9FFFH
            MOV     a,#40h              ;选中十位 LED
            MOVX    @DPTR ,A
            MOV     A,VBDIGIT
            MOV     DPTR,#TABLE
            MOVC    A,@A+DPTR
            MOV     DPTR,#8FFFH
            MOVX    @DPTR,A
            MOV     DPTR,#9FFFH
            MOV     A,#40H
            MOVX    @DPTR ,A
            LCALL   DELY10MS
            MOV     DPTR,#9FFFH
            MOV     A,#0FFH
```




```
            MOVX    @DPTR ,A
            MOV     DPTR,#9FFFH
            MOV     A,#80H              ;选中个位 LED
            MOVX    @DPTR ,A
            MOV     A,VADIGIT
            MOV     DPTR,#TABLE
            MOVC    A,@A+DPTR
            MOV     DPTR,#8FFFH
            MOVX    @DPTR,A
            MOV     DPTR,#9FFFH
            MOV     A,#80H
            MOVX    @DPTR ,A
            LCALL   DELY10MS
            MOV     DPTR,#9FFFH
            MOV     A,#00H
            MOVX    @DPTR ,A
            DJNZ    R0, LOOP2
            INC     COUNT
            MOV     A,COUNT
            CJNE    A,#60,LOOP1
            LJMP    LOOP
DELY10MS:   MOV     R6,#20
D1:         MOV     R7,#248
            DJNZ    R7,$
            DJNZ    R6,D1
            RET
TABLE:      DB      3FH,06H,5BH,4FH,66H,6DH,7DH,07H,7FH,6FH
                END
```

（2）C 语言程序

```
#include<reg51.h>
#include<absacc.h>
#include<intrins.h>
#define uchar unsigned char
#define uint unsigned int
uchar showdata[8]={1,2,3,4,5,6,7,8};
//*************数码管编码***共阴 0-F-无-负号**********************************
uchar showcode[18]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,
                    0x77,0x7C,0x39,0x5E,0x79,0x71,0x00,0x40};
//8255 C 语言接口程序
#define PORTA  &XBYTE[0x8FFF]        //8255A 口地址
#define PORTB  &XBYTE[0x9FFF]        //8255B 口地址
#define PORTC  &XBYTE[0x0AFFF]       //8255C 口地址
#define CADDR  &XBYTE[0x0BFFF]       //8255 控制字地址
void delay10Ms(void)
{
  unsigned char i,j;
```

```
    for(i=20;i>0;i--)
    for(j=248;j>0;j--);
}
void write_8255(uchar xdata * ad_adr,uchar dat)
{
        *ad_adr=dat;
}
void init_8255(void)
{
        write_8255(CADDR,0x81);         //设置 A 口为工作方式 0，输出；C 口低四位设置为输入
                                        //设置 B 口为工作方式 0，输出；C 口高四位设置为输出
        write_8255(PORTA,0x00);         //设置 A 口为 0x00
        write_8255(PORTB,0x00);         //设置 B 口为 0x00
        write_8255(PORTC,0xff);         //设置 C 口为 0xff
}
void main(void)
{char i,Time=0;
        init_8255();
             while（1）
        {
              for(i=0;i<50;i++)
             {
              showdata[1]=Time/10;
              showdata[0]=Time%10;
                write_8255(PORTB,0x00);                             //设置 B 口
                write_8255(PORTA,showcode[showdata[0]]);            //设置 A 口
                write_8255(PORTB,0x80);                             //设置 B 口
                                                                    //数码管 2
              delay10Ms();
                write_8255(PORTB,0x00);                             //设置 B 口
                write_8255(PORTA,showcode[showdata[1]]);            //设置 A 口
                write_8255(PORTB,0x40);                             //设置 B 口
                                                                    //设置 B 口
              delay10Ms();
              }
                     Time++;
                     if(Time==60)Time=0;
              }
}
```

▷▷ 16.4 案例笔记

▷▷▷ 16.4.1 51 单片机定时计数器工作方式总结

定时和计数实质都是对脉冲的计数，只是被计脉冲的来源不同，定时方式的计数脉冲来

源于时钟，计数方式的计数脉冲来源于外部，定时方式的计数初值和计数脉冲周期有关，计数方式的和计数脉冲的个数有关。

定时/计数器应用非常广泛，如定时采样、时间测量、产生音响、作脉冲源、制作日历时钟、测量波形的频率和占空比、检测电动机转速等，因此应掌握其用法。

在 51 单片机内部含有两个 16 位定时/计数器：T0 和 T1，它们由振荡器分频输入电路、外部计数脉冲输入电路、计数脉冲选择电路、计数启停电路、加 1 计数器和中断标志等组成。其核心是一个加 1 计数器，每输入一个脉冲，计数值加 1，当计数到计数器全为 1 时，再输入一个脉冲就使计数值回零，同时从最高位溢出一个脉冲使控制寄存器 TCON 的 TFX（X=0 或 1）位置 1，作为计数器的溢出中断标志。

（1）计数器初值的计算

设需要计数器计数的个数为 X，计数初值为 C，由此可得出如下计算计数器初值的公式：

$$C=M-X$$

式中，M 为定时/计数器的模值，该值和计数器的工作方式有关（为 2^n，n 为工作方式决定的定时/计数器位数）。

（2）定时器初值的计算

在定时模式下，计数器对单片机振荡频率 f_{osc} 经 12 分频后的机器周期进行加 1 计数，用 X 表示计数个数，M 表示模，C 表示定时器初值，T 表示机器周期（12 倍时钟周期），则 $T=12/f_{osc}$，因此，定时时间 t 的计算公式为

$$t = XT = (M - C)T$$

定时器初值公式为

$$C = M - t/T$$

将 $M = 2^n$ 代入上式得

$$C = 2^n - t/T$$

（3）四种方式最大定时时间

由于定时时间为

$$t = (M - C)T = (2^n - C)T$$

因此上式中令 $C = 0$ 则可得各方式最大定时时间。设单片机脉冲频率为 f_{osc}=12MHz，计数周期为 1μs，则得到各方式最大定时时间如下。

方式 0：

$$t_{max} = (2^n - C)T = 2^{13} \times 1\mu s = 8.192ms$$

方式 1：

$$t_{max} = (2^n - C)T = 2^{16} \times 1\mu s = 65.536ms$$

方式 2 和 3：

$$t_{max} = (2^n - C)T = 2^8 \times 1\mu s = 0.256ms$$

【例 16-1】 晶振频率分别为 6MHz 和 12MHz 时，方式 0～方式 3 的最长定时时间各为多少？

解：各方式的最大定时时间见表 16-1。

表 16-1 各方式的最大定时时间

最长定时时间	6MHz	12MHz
方式 0	16.384ms	8192μs
方式 1	131.072ms	65.536ms
方式 2 和 3	512μs	256μs

由于本案例要求定时 1s，按照四种方式的最大定时时间，每种方式单次都不可能完成本案例任务。因此选用单次定时最长的方式 1，为便于取整运算，单次定时 50ms，设置软件计数器计数 20 次，可定时 50ms×20=1s。

此外，若定时时间超过各方式单次定时时间，将两个定时器中的一个（如 T0）设置为定时方式，另一个（T1）设置成计数方式；将定时器 T0 计满产生的输出，通过一根 I/O 口线（如 P1.0）连接到计数器 T1 的计数脉冲输入端 P3.5，从而实现更长时间的定时。

▷▷▷ 16.4.2 C51 精确延时

在 C51 中要实现对时间的精确延时有以下几种方法：

1）对于延时很短的，要求在μs 级的，采用“_nop_”函数，这个函数相当汇编 NOP 指令，延时几微秒，就插入一个这样的函数。

2）对于延时比较长的，要求大于 10μs 时，采用 C51 中的循环语句来实现。对于要求精确延时时间更长的（ms 级），这时就要采用循环嵌套的方法来实现。

在选择 C51 中循环语句时，要注意以下几个问题：

1）定义的 C51 中循环变量，尽量采用无符号字符型变量。

2）在 for 循环语句中，尽量采用变量减减来做循环。

3）在 do-while、while 语句中，循环体内变量也采用减减方法。这因为在 C51 编译器中，对于不同的循环方法，是采用不同的指令来完成的。下面举例说明：

① unsigned char I;

```
for(i=0;i<255;i++);
```

② unsigned char I;

```
for(i=255;i>0;i--);
```

其中，第二个循环语句 C51 编译后，就用 DJNZ 指令来完成，相当于如下指令：

```
MOV   09H，#0FFH
LOOP：DJNZ   09H，LOOP
```

指令相当简洁，也很好地计算出精确的延时时间。

同样对 do-while、while 循环语句中，也是如此，例如：

```
unsigned char n;
n=255;
```

```
do{n--}
while(n);
```

或

```
n=255;
while(n)
{n--};
```

4）对于循环语句同样可以采用 for、do-while、while 结构来完成，每个循环体内的变量仍然采用无符号字符变量。

①

```
unsigned char i,j
for(i=255;i>0;i--)
for(j=255;j>0;j--);
```

②

```
unsigned char i,j
i=255;
do{j=255;
    do{j--}
    while(j);
    i--;
  }
while(i);
```

③

```
unsigned char i,j
i=255;
while(i)
{j=255;
  while(j)
  {j--};
  i--;
}
```

这三种方法都是用 DJNZ 指令嵌套实现循环的，由 C51 编译器用下面的指令组合来完成的：

```
MOV   R7，#0FFH
LOOP2：MOV   R6，#0FFH
LOOP1：DJNZ   R6，LOOP1
DJNZ   R7，LOOP2
```

这些指令的组合在汇编语言中采用 DJNZ 指令来做延时用，因此它的时间精确计算也很简单，例如，变量 i 的初值为 m，变量 j 的初值为 n 时，则总延时时间为 $m(nT+T)$，其中 T 为 DJNZ 指令执行时间。

本案例中，C51 的纯软件延时程序如下：

```
void delay10Ms(void)
{
   unsigned char i,j;
   for(i=20;i>0;i--)
   for(j=248;j>0;j--);
}
```



▷▷▷ 16.4.3 中断与定时/计数器综合应用

定时/计数功能与中断一样，都是单片机的常用功能。两者常常同时使用。在这类程序的编制过程中，要注意这样几个问题：

1）选择合适的中断和定时/计数方式。如外部中断是采用电平触发还是脉冲下降沿触发；定时/计数是用自动重装方式还是每次定时结束后用软件重装。

2）确定定时结束的判别方法，用中断还是查询。如采用中断，与其他中断的优先级如何确定，是否会影响系统功能。

3）正确初始化。

4）合理分配控制功能。

在单片机控制系统中，外部中断的使用非常重要，通过它可以中断 CPU 的运行，转去处理更为紧迫的外部事务，如报警、电源掉电保护等。51 单片机仅提供了两个外部中断源（$\overline{INT0}$/$\overline{INT1}$），在实际控制系统中可能出现多个外部中断，因此有必要对外部中断源进行扩展，详见 16.4.4 节。

▷▷▷ 16.4.4 实用技巧之扩展中断源

51 单片机内部有两个 16 位的定时/计数器，当它们发生溢出时，会向 CPU 发出中断请求。因此，可以把内部不使用的定时/计数器出借给外部中断使用，扩展出一个（或两个）外部中断源。

【例 16-2】 利用定时计数器 T1 在方式 2 下扩展中断源。

分析：可将 T1 设置为计数方式，初值为最大值。一旦外部信号从计数器引脚输入一个负跳变信号，计数器加 1，产生溢出中断，从而可以转去处理该外部中断源的请求。因此，可以把 P3.5 作为外部中断请求输入线，而溢出标志位相当于外部中断请求标志位。程序如下：

```
MOV     TMOD,#60H      ;T1 方式 2
MOV     TH1,#0FFH
MOV     TL1,#0FFH      ;置初值
SETB    TR1            ;启动计数器 T1
SETB    EA             ;微处理器中断开放
SETB    ET1            ;允许 T1 中断
```

当 P3.5 上的信号发生负跳变时，TL1 加 1 溢出，TF1 置 1，向微处理器发出中断申请。同时 TH1 的内容送 TL1，即恢复计数器初值 0FFH。这样 P3.5 引脚上的每次负跳变都将 TF1 置 1，向微处理器发出中断请求，微处理器响应中断请求，转去执行外部中断服务程序，其入口地址为 001BH，此时 P3.5 相当于边沿触发的外中断源输入线。

案例17

 频率显示器

17.1 案例任务

改变 555 电路所接的滑动变阻器进而改变 555 电路的输出频率，将频率值在 LED 上显示出来。通过定时 1s 对 555 电路的输入脉冲计数获得频率值。

17.2 案例要点

（1）555 定时器的内部结构及原理

555 定时器是一种将模拟功能与逻辑功能结合在一起的混合集成电路，应用遍及电子产品的各个领域，在其外部配上少量阻容元件，便能构成多谐振荡器、单稳态触发器、施密特触发器等电路。

结构图如图 17-1 所示，它主要由三个阻值为 5kΩ的电阻组成的分压器、两个高精度的电压比较器 C_1 和 C_2、基本 RS 触发器以及一个作为放电通路的晶体管 VT 组成。为了提高电路的驱动能力，在输出级又增加了一个非门 G。

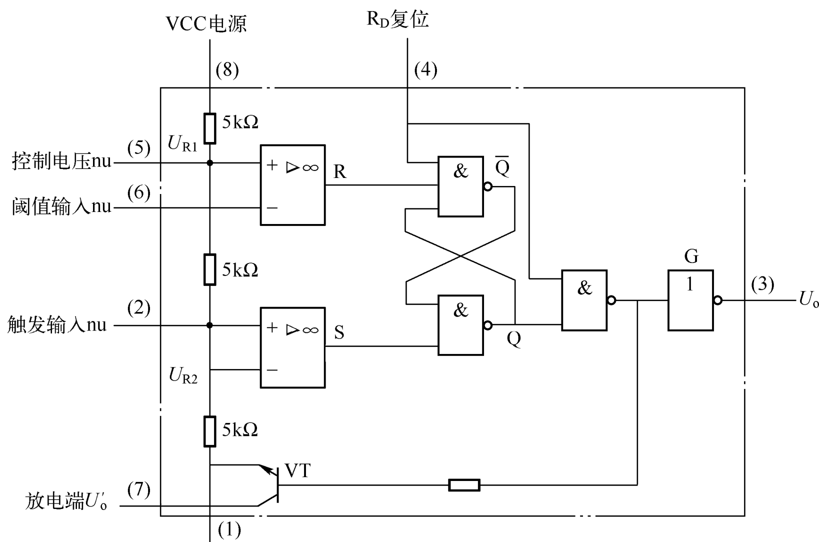


图 17-1 555 电路内部结构

（2）定时/计数器测量脉冲个数的原理

定时计数器 1 工作于方式 1，用于定时 50ms，20 次后为 1s；定时计数器 0 也工作于方

式 1，用于对脉冲个数进行计数；在定时 1s 的中断中，读出定时计数器 0 中的寄存器数据，显示在 LED 上。

（3）定时/计数器测脉冲的硬件电路连接及驱动程序编写

17.3 案例设计

17.3.1 硬件电路

频率显示器电路如图 17-2 所示。首先将 555 电路的 TRIG 引脚和 THOLD 引脚连到一起构成施密特触发器；然后将施密特触发器的反相输出端 DISCHG 经过 *RC* 积分电路(R_4 和 E_5)接回到输入端 TRIG 和 THOLD 引脚，便可以形成多谐振荡器，高电平持续时间为

$$t_1 = (R_3 + R_4)E_5 \ln 2$$

低电平持续时间为

$$t_2 = R_4 E_5 \ln 2$$

振荡周期为

$$T = t_1 + t_2 = (R_3 + 2R_4)E_5 \ln 2$$

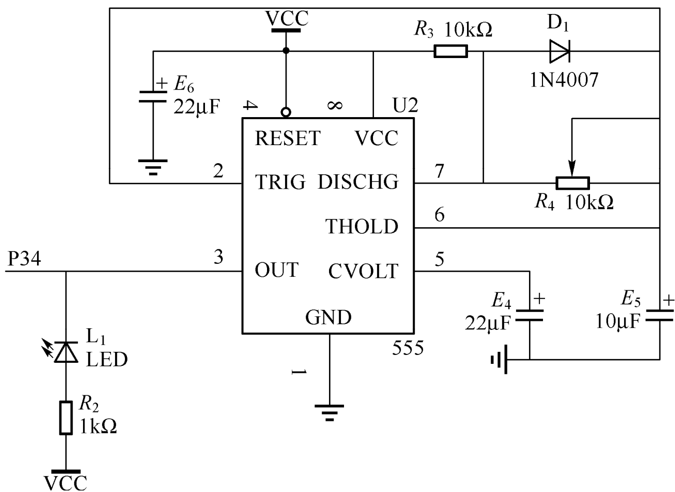


图 17-2 频率显示器电路

17.3.2 软件代码

（1）汇编程序

```
PORTA      EQU     8FFFH    ;8255A 口地址
PORTB      EQU     9FFFH    ;8255B 口地址
PORTC      EQU     0AFFFH   ;8255C 口地址
CADDR      EQU     0BFFFH   ;8255 控制字地址
TIMCNT1    EQU     7AH
```

```
SHOWFRE     EQU     7BH
            ORG     0000H
            AJMP    MAIN
            ORG     001BH
            AJMP    TIME1
            ORG     0100H
MAIN:       MOV     A,#81H
            MOV     DPTR,#CADDR
            MOVX    @DPTR,A             ;设置 8255 工作方式
            MOV     TMOD,#15H
            MOV     TH1,#3CH
            MOV     TL1,#0B0H
            MOV     TH0,#00H
            MOV     TL0,#00H
            SETB    TR1
            SETB    TR0
            SETB    ET1
            SETB    EA
            MOV     TIMCNT1,#20
LOOP:       MOV     A,R3
            MOV     B,#100
            DIV     AB
            MOV     R2,A                ;R2 为百位
            MOV     A,B
            MOV     B,#10
            DIV     AB
            MOV     R1,A                ;R1 为十位
            MOV     R0,B                ;R0 为个位
            MOV     A,#80H
            MOV     DPTR,#PORTB
            MOVX    @DPTR,A             ;置个位数码管有效
            MOV     DPTR,#ZXM
            MOV     A,R0
            MOVC    A,@A+DPTR           ;设置个位显示码
            MOV     DPTR,#PORTA
            MOVX    @DPTR,A
            CALL    DELAY
            MOV     DPTR,#ZXM
            MOV     A,R1
            MOVC    A,@A+DPTR           ;设置十位显示码
            MOV     DPTR,#PORTA
            MOVX    @DPTR,A
            MOV     A,#40H              ;置十位数码管有效
            MOV     DPTR,#PORTB
            MOVX    @DPTR,A
            CALL    DELAY
```

```
            MOV     DPTR,#ZXM
            MOV     A,R2
            MOVC    A,@A+DPTR       ;设置百位显示码
            MOV     DPTR,#PORTA
            MOVX    @DPTR,A
            MOV     A,#020H         ;置百位数码管有效
            MOV     DPTR,#PORTB
            MOVX    @DPTR,A
            CALL    DELAY
            JMP     LOOP
DELAY:      MOV     R6,#20          ;延时子程序
LOOP1:      MOV     R7,#255
            DJNZ    R7,$
            DJNZ    R6,LOOP1
            RET
;实验板上的 7 段数码管显示 0～9 数字
/*********************************************/
/*          定时器 1 中断服务                 */
/*********************************************/
TIME1:      MOV     TH1,#3CH
            MOV     TL1,#0B0H
            DJNZ    TIMCNT1,RETURN0
            MOV     TIMCNT1,#20
            MOV     R3,TL0
            MOV     TH0,#00H
            MOV     TL0,#00H
RETURN0:    RETI
;实验板上的 7 段数码管 0～9 数字的共阴显示代码
ZXM:        DB      3FH,06H,5BH,4FH,66H,6DH,7DH,07H,7FH,6FH
    END
```

（2）C 语言程序

```
unsigned char xdata porta _at_ 0x8fff;
unsigned char xdata portb _at_ 0x9fff;
unsigned char xdata portc _at_ 0xafff;
unsigned char xdata caddr _at_ 0xbfff;
unsigned char xdata *a _at_ 0x0060;
unsigned char xdata *b _at_ 0x0065;
unsigned char xdata *c _at_ 0x0070;
unsigned char num[10]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};
#include<AT89X51.h>
unsigned char m=0;
unsigned char count=0;
unsigned char n0=0;
unsigned char n1=0;
```

```
unsigned char n2=0;
unsigned char temp=0;
unsigned int n;
void main (void)
{
 c=&caddr;
 *c=0x80;
 a=&porta;
 b=&portb;
 TMOD=0x15;   //T0  计数，方式 1；T1 定时，方式 1
 TH1=0x3c;
 TL1=0xb0;
 TH0=0x00;
 TL0=0x00;
 EA=1;
 ET1=1;
 TR0=1;
 TR1=1;
 while(1)
        {
         n2=count/100;   //百位数字
         temp=count%100;
         n1=temp/10;
         n0=temp%10;
         *b=0x80;
         *a=num[n0];
         for (n=0;n<1000;n++);
         *b=0x40;
         *a=num[n1];
         for (n=0;n<1000;n++);
         *b=0x20;
         *a=num[n2];
         for (n=0;n<1000;n++);
        }
}
timer1() interrupt 3
        {
         TH1=0x3c;
         TL1=0xb0;
         m++;
         if(m==20)
             {
              count=TL0;
              TL0=0;
              m=0;
             }
         }
```

17.4 案例笔记

17.4.1 频率的测量方法

一般来说，频率测量所需要的接口简单，占用资源少，可只用一路计数器直接计数或者用中断输入接口触发中断，然后在中断服务程序中再对脉冲进行计数。另外，对频率信号的其他参数测量也有重要的意义，如周期、高低电平等。

1．测频法

测频法是指在一定的时间内直接对信号的边沿触发或电平触发进行计数，如图 17-3 所示，也可以称为计数法。被测信号是一串计数脉冲（实际中应通过放大整形得到），将它加到闸门的一个输入端，闸门由门控信号来控制其关闭时间。将单位门控时间内计得的脉冲送至处理器处理，再经显示器显示。图 17-3 中，定义被测信号的频率为 f_x，闸门开启时间为 T_w，在这段时间内所计量得到的脉冲个数为 N_x，则被测信号频率可以表示为

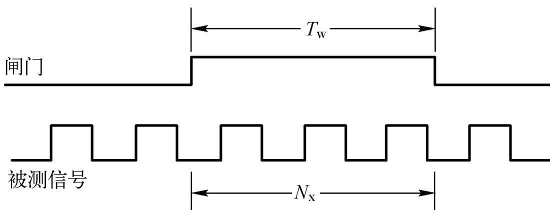


图 17-3 测频法

$$f_x = N_x/T_w \tag{17-1}$$

不难看出，采用计数的测频方法的测量误差，一方面取决于闸门时间 T 是否准确，即由晶振提供的标准频率的准确度；另一方面取决于计数器计得的数是否准确。所以，计数测频方法的误差主要有两项，即标准频率误差和计数值误差。在测量高频时，计数值误差引起的测频误差相对较小，所以这种方法比较适合于高频信号的测量。但测低频时，由于计数值误差产生的测频误差非常大，所以不宜采用计数测频方法。

此外，从公式（17-1）中可以看出，要得到频率 f_x，必须知道 T_w 和 N_x，常用的方法是已知其中的一个量，然后对另外一个量进行测量。例如，将 51 单片机中的一个定时器用于得到标准时间的闸门信号 T_w，用外部中断或端口捕获的方法接入信号，对交变信号进行计数，从而得到 N_x。举例来说，如果 T_w 为 1s，则得到的频率即为脉冲个数 N_x。

2．测周法

在前文提到，在对低频信号进行测量时，如果还采用测频法，会导致由计数值引起的巨大误差。因此，在低频时通常使用测周法，即利用信号的一个周期作为时间闸门信号，在这个信号周期对单片机内部的已知脉冲进行计数。如图 17-4 所示，基准信号的周期为 T_s，被测信号的周期为 T_x，则在被测信号的一个周期 T_x 内，记录基准信号的周期数为 N_s，得到被测信号的频率为

$$f_x = 1/T_x = 1/(N_sT_s) \tag{17-2}$$

从公式（17-2）中可以看出，要得到频率 f_x，应知道 T_s 和 N_s。例如，将 51 单片机中的一个定时器用于内部计数，其每计一次的周期时间即为基准信号的周期 T_s（其值由单片机的晶振和指令运行周期决定），用外部中断或端口捕获的方法接入信号后，在一个被测信号的周期开始和周期结束分别触发，在这个过程中得到的脉冲个数即为 N_x。

通常情况下当信号频率在 1kHz 以下时，可以采用测周法；当信号频率在 1kHz 以上时，可以采用测频法。当然在实际调试应用时可以根据实际情况（如单片机捕获能力、晶振偏差等）进行微调修改。

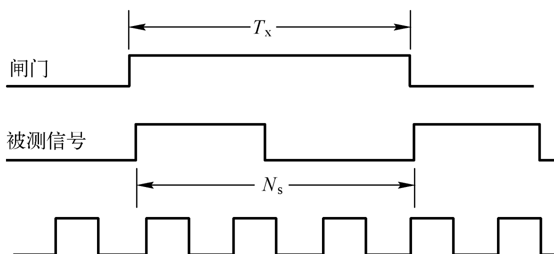


图 17-4　测周法

上述所介绍的这两种测频方案，各有优缺点，这两种方案有一个中间临界值的问题，或者，到底所测频率多大时可交换这两种测频方法，这是测频系统的一个关键点。

17.4.2　利用门控制位 GATE 测量脉冲宽度

GATE1 可使定时器/计数器 T1 的启动计数受 $\overline{\text{INT1}}$ 的控制，可测量引脚 $\overline{\text{INT1}}$（P3.3）上正脉冲的宽度（机器周期数），如图 17-5 所示。

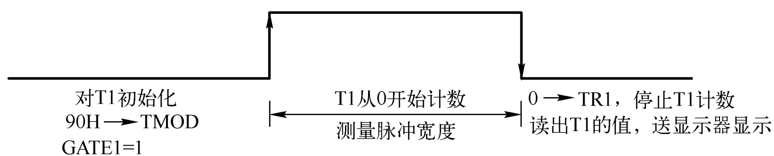


图 17-5　利用 GATE 测量脉冲宽度

相关程序如下：

```
        ORG     0000H
        AJMP    MAIN            ;复位入口转主程序
        ORG     0100H
MAIN:   MOV     SP,#60H
        MOV     TMOD,#90H       ;T1 为方式 1 定时控制字
        MOV     TL1,#00H
        MOV     TH1,#00H
LOOP0:  JB      P3.3,LOOP0      ;INT1 高，则循环
        SETB    TR1             ;如 INT1 为低，则启动 T1
LOOP1:  JNB     P3.3,LOOP1      ;INT1 低，则循环
LOOP2:  JB      P3.3,LOOP2      ;INT1 高，则循环
        CLR     TR1             ;停止 T1 计数
        MOV     A,TL1           ;T1 计数值送 A
将 A 中的 T1 计数值送显示缓冲区转换成显示的代码
LOOP3:  LCALL   DIR             ;调用显示子程序 DIR
        AJMP    LOOP0
```

案例18



电压监控器



18.1 案例任务

将 ADC0809 第 IN0 路输入与滑动变阻器通过跳线短接，当改变滑动变阻器阻值时，通过分压原理分得的电压值在 LED 上显示出来。

18.2 案例要点

（1）掌握 ADC0809 技术指标

1）8 路 8 位 A-D 转换器，即分辨率为 8 位。

2）具有转换启停控制端。

3）转换时间为 100μs。

4）单个＋5V 电源供电。

5）模拟输入电压范围为 0～＋5V，无需零点和满刻度校准。

6）工作温度范围为-40～＋85℃。

7）低功耗，约 15mW。

（2）ADC0809 引脚功能和与 51 单片机的接口电路，及端口寻址

端口寻址主要由 ADDA、ADDB、ADDC 三个引脚与单片机连接电路决定。

（3）ADC0809 的三种驱动程序编写方法：中断、查询、软件延时

1）中断方式。将转换结束标志引脚接到 51 单片机的中断申请引脚(如 INT0、INT1)。当转换结束时，即提出中断申请，51 单片机响应后，在中断服务程序中读取数据。这种方法使 A-D 转换器与 51 单片机的工作同时进行，因而节省时间，常用于实时性要求比较高或多参数的数据采集系统。

2）查询方式。把转换结束信号经三态门送到 51 单片机的数据总线或 I/O 接口的某一位上。51 单片机向 A-D 转换器发出启动信号后，便开始查询 A-D 转换是否结束，一旦查询到 A-D 转换结束，就读出结果数据。这种方法的程序设计比较简单，不过占用较多的单片机时间。

3）软件延时方法。51 单片机启动 A-D 转换后，根据转换芯片完成转换所需要的时间，调用一段延时程序（为确保 A-D 转换已完成，通常程序延时时间略大于 A-D 转换过程所需的时间）。延时程序执行完毕后，A-D 转换也已完成，这时即可读出结果数据。这种方法无需检测 A-D 转换结束标志引脚，但在以上三种方法中占用最多的单片机时间。



18.3 案例设计

18.3.1 硬件电路

电压监控器电路如图 18-1 所示。

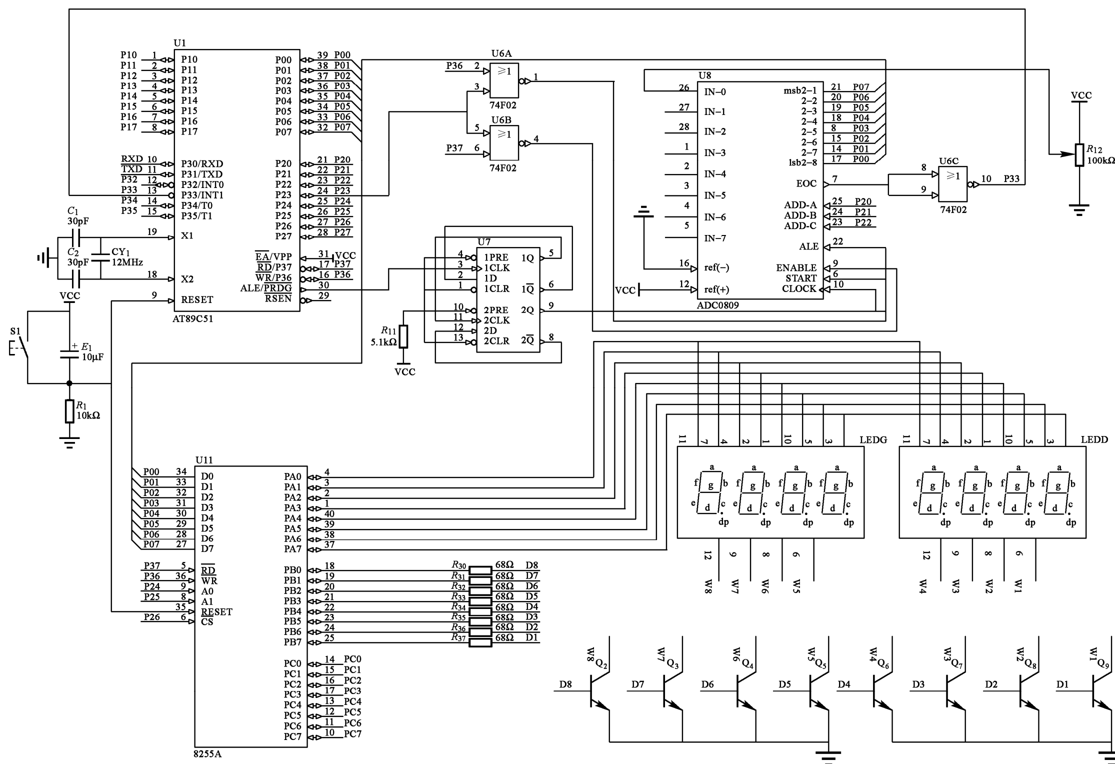


图 18-1 电压监控器电路

18.3.2 软件代码

1．中断方式读入

（1）汇编程序

```
ADDR_A      EQU     08FFFH          ;8255A 口地址为 8FFFH
ADDR_B      EQU     09FFFH          ;8255B 口地址为 9FFFH
ADDR_C      EQU     0AFFFH          ;8255C 口地址为 0AFFFH
ADDR_K      EQU     0BFFFH
DATA_AD     EQU     30H
_ST         EQU     P3.6
_OE         EQU     P3.7
_EOC        EQU     P3.3
            ORG     0000H
            SJMP    MAIN
            ORG     0013H
            AJMP    PINT1
```



```
        ORG     0030H
MAIN:   SETB    EA              ;允许总中断
        SETB    EX1             ;开 INT1 中断
        SETB    IT1             ;负跳变触发
        MOV     DPTR,#ADDR_K
        MOV     A,#80H
        MOVX    @DPTR,A
        MOV     DPTR,#ADDR_C
        MOV     A,#0FFH
        MOVX    @DPTR,A
        SETB    _OE             ;初始化时使_ST 与_OE 全为低电平
        SETB    _ST
        ANL     P2,#0F0H        ;P2.3 为低选中 0809，
                                P2.2、P2.1、P2.0 送通道号 00
        MOV     DPTR,#0F0FFH
        MOVX    @DPTR,A
TRANS:  MOV     DPTR,#0F0FFH
        MOVX    @DPTR,A
        MOV     A,DATA_AD
        MOV     B,#51           ;近似转换得电压整数值
        DIV     AB
        MOV     R3,A
        XCH     A,B
        MOV     B,#5
        DIV     AB
        MOV     R2,A            ;近似得电压 0.1V 数值
        MOV     R1,B            ;近似得电压 0.01V 数值
DISPLAY: MOV    DPTR,#ADDR_B
        MOV     A,#040H         ;选中显示 0.01V 的 LED
        MOVX    @DPTR,A
        MOV     A,R1
        MOV     DPTR,#ZXM
        MOVC    A,@A+DPTR
        MOV     DPTR,#ADDR_A
        MOVX    @DPTR,A
        ACALL   DELAY
        MOV     DPTR,#ADDR_B
        MOV     A,#020H         ;选中显示 0.1V 的 LED
        MOVX    @DPTR,A
        MOV     A,R2
        MOV     DPTR,#ZXM
        MOVC    A,@A+DPTR
        MOV     DPTR,#ADDR_A
        MOVX    @DPTR,A
        CALL    DELAY
        MOV     DPTR,#ADDR_B
        MOV     A,#010H         ;选中显示 1V 的 LED
        MOVX    @DPTR,A
        MOV     A,R3
        ADD     A,#11           ;加上小数点
        MOV     DPTR,#ZXM
        MOVC    A,@A+DPTR
        MOV     DPTR,#ADDR_A
        MOVX    @DPTR,A
        ACALL   DELAY
        SJMP    TRANS
DELAY:  MOV     R5,#02H
```

```
L1:         MOV     R6,#0E0H
L2:         DJNZ    R6,L2
            DJNZ    R5,L1
            RET
PINT1:      SETB    P1.3
            MOV     DPTR,#0F0FFH
            MOVX    A,@DPTR
            MOV     DATA_AD,A
                    RETI
ZXM:        DB      3fh,06h,5bh,4fh,66h,6dh,7dh,07h,7fh,6fh,3eH
            DB      0bfh,86h,0dbh,0cfh,0e6h,0edh,0fdh,087h,0ffh,0efh
                    END
```

（2）C 语言程序

```
#include<reg51.h>
#include<absacc.h>
#include<intrins.h>
#define uchar unsigned char
#define uint unsigned int
#define IN0(&XBYTE[0x0f0FF])              //ADC08090 通道地址
sbit ST_0809    =    P3^6;                //开始 A-D 转换
sbit EN_0809    =    P3^7;                //使转换数据输出，即读
sbit EOC_0809   =    P3^3;                //转换完毕标置位

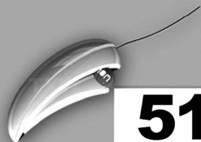
#define PORTA   &XBYTE[0x8FFF]            //8255A 口地址
#define PORTB   &XBYTE[0x9FFF]            //8255B 口地址
#define PORTC   &XBYTE[0x0AFFF]           //8255C 口地址
#define CADDR   &XBYTE[0x0BFFF]           //8255 控制字地址

#define DelayTime 100
uchar showdata[8]={1,2,3,4,5,6,7,8};
/*************精确延时函数**************************************/
//延时(14+6*i)μs，i 最小为 0，最大为 255，即 1544μs
void Delayus(uchar i)
{
    while(i--);
}
void write_8255(uchar xdata * ad_adr,uchar dat)
{
    *ad_adr=dat;
}
void init_8255(void)
{
    write_8255(CADDR,0x81);        //设置 A 口为工作方式 0，输出；C 口低四位设置为输入
                                   //设置 B 口为工作方式 0，输出；C 口高四位设置为输出
    write_8255(PORTA,0x00);        //设置 A 口为 0x00
    write_8255(PORTB,0x00);        //设置 B 口为 0x00
    write_8255(PORTC,0xff);        //设置 C 口为 0xff;
```

```
}
//*************数码管编码***共阴 0-F-无-负号*****************
uchar showcode[18]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,
                    0x7C,0x39,0x5E,0x79,0x71,0x00,0x40};
                   void show8(void)
{uchar times=0;
    for(times=50;times;times--)
    {                                                      //数码管 1
        write_8255(PORTB,0x00);                            //设置 B 口
        write_8255(PORTA,showcode[showdata[0]]);           //设置 A 口
        write_8255(PORTB,0x80);                            //设置 B 口
        Delayus(DelayTime);                                //数码管 2
        write_8255(PORTB,0x00);                            //设置 B 口
        write_8255(PORTA,showcode[showdata[1]]|0x80);      //设置 A 口
        write_8255(PORTB,0x40);                            //设置 B 口
        Delayus(DelayTime);
    }
}
void Init_Int1(void)                    //外部中断 0
{
    IT1=1;                              //负边沿触发中断
    EX1=1;                              //打开外部中断 0
}
void int1(void) interrupt 2
{uchar addata=0;
    EX1=0;                              //关外部中断 1
   //*IN0=0xff;
    addata=(*IN0)*0.196;
    showdata[0]=addata%10;
    showdata[1]=addata/10;
    EX1=1;
}
void main(void)
{
    init_8255();
    Init_Int1();
    EA=1;                               //打开全局中断
    while(1)
    {
        *IN0=0xff;
        show8();
    }
}
```

2．查询方式读入

（1）汇编程序




```
ADDR_A      EQU     08FFFH          ;8255A 口地址为 8FFFH
ADDR_B      EQU     09FFFH          ;8255B 口地址为 9FFFH
ADDR_C      EQU     0AFFFH          ;8255C 口地址为 0AFFFH
ADDR_K      EQU     0BFFFH
_ST         EQU     P3.6
_OE         EQU     P3.7
_EOC        EQU     P3.3
            ORG     0000H
            SJMP    MAIN
            ORG     0030H
MAIN:       MOV     DPTR,#ADDR_K
            MOV     A,#80H
            MOVX    @DPTR,A
            MOV     DPTR,#ADDR_C
            MOV     A,#0FFH
            MOVX    @DPTR,A
            SETB    _OE             ;初始化时使_ST 与_OE 全为低电平
            SETB    _ST
            ANL     P2,#0F0H        ;P2.3 为低选中 0809，P2.2、P2.1、
                                    P2.0 送通道号 00
LOOP:       MOV     DPTR,#0f0FFH    ;启动 A-D 转换
            MOVX    @DPTR,A
WAIT:       JB      _EOC,WAIT       ;等待转换结束
READ:       MOVX    A,@DPTR
TRANS:      MOV     B,#51           ;近似转换得电压整数值
            DIV     AB
            MOV     R3,A
            XCH     A,B
            MOV     B,#5
            DIV     AB
            MOV     R2,A            ;近似得电压 0.1V 数值
            MOV     R1,B            ;近似得电压 0.01V 数值
DISPLAY:    MOV     DPTR,#ADDR_B
            MOV     A,#040H         ;选中显示 0.01V 的 LED
            MOVX    @DPTR,A
            MOV     A,R1
            MOV     DPTR,#ZXM
            MOVC    A,@A+DPTR
            MOV     DPTR,#ADDR_A
            MOVX    @DPTR,A
            ACALL   DELAY
            MOV     DPTR,#ADDR_B
            MOV     A,#020H         ;选中显示 0.1V 的 LED
            MOVX    @DPTR,A
            MOV     A,R2
            MOV     DPTR,#ZXM
            MOVC    A,@A+DPTR
            MOV     DPTR,#ADDR_A
            MOVX    @DPTR,A
            CALL    DELAY
            MOV     DPTR,#ADDR_B
            MOV     A,#010H         ;选中显示 1V 的 LED
            MOVX    @DPTR,A
            MOV     A,R3
            ADD     A,#11           ;加上小数点
            MOV     DPTR,#ZXM
```

```
            MOVC    A,@A+DPTR
            MOV     DPTR,#ADDR_A
            MOVX    @DPTR,A
            ACALL   DELAY
            SJMP    LOOP
DELAY:      MOV     R5,#02H
L1:         MOV     R6,#0E0H
L2:         DJNZ    R6,L2
            DJNZ    R5,L1
                    RET
ZXM:        DB      3fh,06h,5bh,4fh,66h,6dh,7dh,07h,7fh,6fh,3eH
            DB      0bfh,86h,0dbh,0cfh,0e6h,0edh,0fdh,087h,0ffh,0efh
                    END
```

（2）C 语言程序

```
#include<reg51.h>
#include<absacc.h>
#include<intrins.h>
#define uchar unsigned char
#define uint unsigned int
uchar showdata[8]={1,2,3,4,5,6,7,8};
/**************精确延时函数**************************************/
//延时(14+6*i)μs，i 最小为 0，最大为 255，即 1544μs
void Delayus(uchar i)
{
    while(i--);
}
//8255 C 语言接口程序
#define PORTA  &XBYTE[0x8FFF]        //8255A 口地址
#define PORTB  &XBYTE[0x9FFF]        //8255B 口地址
#define PORTC  &XBYTE[0x0AFFF]       //8255C 口地址
#define CADDR  &XBYTE[0x0BFFF]       //8255 控制字地址
void write_8255(uchar xdata * ad_adr,uchar dat)
{
    *ad_adr=dat;
}
void init_8255(void)
{
    write_8255(CADDR,0x81);      //设置 A 口为工作方式 0，输出；C 口低四位设置为输入
                                 //设置 B 口为工作方式 0，输出；C 口高四位设置为输出
    write_8255(PORTA,0x00);      //设置 A 口为 0x00
    write_8255(PORTB,0x00);      //设置 B 口为 0x00
    write_8255(PORTC,0xff);      //设置 C 口为 0xff
}
//*************数码管编码***共阴 0-F-无-负号********************
uchar showcode[18]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,
                    0x7C,0x39,0x5E,0x79,0x71,0x00,0x40};
#define DelayTime 100
```

```
uchar times=0;
void show8(void)
{
    for(times=50;times;times--)
    {                                                           //数码管 1
        write_8255(PORTB,0x00);                                 //设置 B 口
        write_8255(PORTA,showcode[showdata[0]]);                //设置 A 口
        write_8255(PORTB,0x80);                                 //设置 B 口
        Delayus(DelayTime);
                                                                //数码管 2
        write_8255(PORTB,0x00);                                 //设置 B 口
        write_8255(PORTA,showcode[showdata[1]]|0x80);           //设置 A 口
        write_8255(PORTB,0x40);                                 //设置 B 口
        Delayus(DelayTime);
        write_8255(PORTB,0x00);                                 //设置 B 口
    }
}
#define IN0&XBYTE[0x0f0FF]                                      //ADC08090 通道地址
sbit ST_0809    =    P3^6;                                      //开始 A-D 转换
sbit EN_0809    =    P3^7;                                      //使转换数据输出
sbit EOC_0809   =    P3^3;                                      //转换完毕标置位
uchar read_0809(uchar xdata * ad_adr)
{
    *ad_adr=0xff;
    while(EOC_0809==0);
    return *ad_adr;
}
uchar addata=0;
void san0809(void)
{
    addata=read_0809(IN0)*0.196;
    showdata[0]=addata%10;
    showdata[1]=addata/10;
}
void main(void)
{
    init_8255();
    san0809();
    while(1)
    {
        san0809();
        show8();
    }

}
```

3. 延时方式读入

（1）汇编程序



```
ADDR_A      EQU     08FFFH          ;8255A 口地址为 8FFFH
ADDR_B      EQU     09FFFH          ;8255B 口地址为 9FFFH
ADDR_C      EQU     0AFFFH          ;8255C 口地址为 0AFFFH
ADDR_K      EQU     0BFFFH
_ST         EQU     P3.6
_OE         EQU     P3.7
_EOC        EQU     P3.3
            ORG     0000H
            SJMP    MAIN
            ORG     0030H
MAIN:       MOV     DPTR,#ADDR_K
            MOV     A,#80H
            MOVX    @DPTR,A
            MOV     DPTR,#ADDR_C
            MOV     A,#0FFH
            MOVX    @DPTR,A
            SETB    _OE             ;初始化时使_ST 与_OE 全为低电平
            SETB    _ST
            ANL     P2,#0F0H        ;P2.3 为低选中 0809，P2.2、P2.1、
                                    P2.0 送通道号 00
LOOP:       MOV     DPTR,#0f0FFH    ;启动 A-D 转换
            MOVX    @DPTR,A
            ACALL   DLY1            ;延时转换结束
READ:       MOVX    A,@DPTR
TRANS:      MOV     B,#51           ;近似转换得电压整数值
            DIV     AB
            MOV     R3,A
            XCH     A,B
            MOV     B,#5
            DIV     AB
            MOV     R2,A            ;近似得电压 0.1V 数值
            MOV     R1,B            ;近似得电压 0.01V 数值
DISPLAY:    MOV     DPTR,#ADDR_B
            MOV     A,#040H         ;选中显示 0.01V 的 LED
            MOVX    @DPTR,A
            MOV     A,R1
            MOV     DPTR,#ZXM
            MOVC    A,@A+DPTR
            MOV     DPTR,#ADDR_A
            MOVX    @DPTR,A
            ACALL   DELAY
            MOV     DPTR,#ADDR_B
            MOV     A,#020H         ;选中显示 0.1V 的 LED
            MOVX    @DPTR,A
            MOV     A,R2
            MOV     DPTR,#ZXM
            MOVC    A,@A+DPTR
            MOV     DPTR,#ADDR_A
            MOVX    @DPTR,A
            CALL    DELAY
            MOV     DPTR,#ADDR_B
            MOV     A,#010H         ;选中显示 1V 的 LED
```

```
                MOVX        @DPTR,A
                MOV         A,R3
                ADD         A,#11                  ;加上小数点
                MOV         DPTR,#ZXM
                MOVC        A,@A+DPTR
                MOV         DPTR,#ADDR_A
                MOVX        @DPTR,A
                ACALL       DELAY
                SJMP        LOOP
DELAY:          MOV         R5,#02H
L1:             MOV         R6,#0E0H
L2:             DJNZ        R6,L2
                DJNZ        R5,L1
                            RET
DLY1:           MOV         R6,#05
DLY2:           NOP
                DJNZ        R6,DLY2
                RET
ZXM:            DB          3fh,06h,5bh,4fh,66h,6dh,7dh,07h,7fh,6fh,3eH
                DB          0bfh,86h,0dbh,0cfh,0e6h,0edh,0fdh,087h,0ffh,0efh
                            END
```

（2）C 语言程序

```
#include<reg51.h>
#include<absacc.h>
#include<intrins.h>
#define uchar unsigned char
#define uint unsigned int
uchar showdata[8]={1,2,3,4,5,6,7,8};
/**************精确延时函数**************************************/
//延时(14+6*i)μs，i 最小为 0，最大为 255，即 1544μs
void Delayus(uchar i)
{
    while(i--);
}
//8255 C 语言接口程序
#define PORTA   &XBYTE[0x8FFF]                 //8255A 口地址
#define PORTB   &XBYTE[0x9FFF]                 //8255B 口地址
#define PORTC   &XBYTE[0x0AFFF]                //8255C 口地址
#define CADDR  &XBYTE[0x0BFFF]                 //8255 控制字地址
void write_8255(uchar xdata * ad_adr,uchar dat)
{
    *ad_adr=dat;
}
void init_8255(void)
{
    write_8255(CADDR,0x81);       //设置 A 口为工作方式 0，输出；C 口低四位设置为输入
                                  //设置 B 口为工作方式 0，输出；C 口高四位设置为输出
    write_8255(PORTA,0x00);       //设置 A 口为 0x00
```



```
        write_8255(PORTB,0x00);         //设置 B 口为 0x00
        write_8255(PORTC,0xff);         //设置 C 口为 0xff
}
//************数码管编码***共阴 0-F-无-负号*******************
uchar showcode[18]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,
                    0x7C,0x39,0x5E,0x79,0x71,0x00,0x40};
#define DelayTime 100
uchar times=0;
void show8(void)
{
        for(times=50;times;times--)
        {                                                               //数码管 1
                write_8255(PORTB,0x00);                                 //设置 B 口
                write_8255(PORTA,showcode[showdata[0]]);                //设置 A 口
                write_8255(PORTB,0x80);                                 //设置 B 口
                Delayus(DelayTime);                                     //数码管 2
                write_8255(PORTB,0x00);                                 //设置 B 口
                write_8255(PORTA,showcode[showdata[1]]|0x80);           //设置 A 口
                write_8255(PORTB,0x40);                                 //设置 B 口
                Delayus(DelayTime);
                write_8255(PORTB,0x00);                                 //设置 B 口
        }
}
#define IN0&XBYTE[0x0f0FF]                      //ADC08090 通道地址
sbit ST_0809    =       P3^6;                   //开始 A-D 转换
sbit EN_0809    =       P3^7;                   //使转换数据输出
sbit EOC_0809   =       P3^3;                   //转换完毕标置位
uchar read_0809(uchar xdata * ad_adr)
{
         *ad_adr=0xff;
         Delayus(50);
         return *ad_adr;
}
uchar addata=0;
void san0809(void)
{
        addata=read_0809(IN0)*0.196;
        showdata[0]=addata%10;
        showdata[1]=addata/10;
}
void main(void)
{
        init_8255();
        san0809();
        while(1)
        {
```

```
            san0809();
            show8();

        }
    }
```

▷▷ 18.4　案例笔记

▷▷▷ 18.4.1　逐次逼近式 A–D 转换

ADC0809 的转换原理为逐次逼近式 A–D 转换。逐次逼近式 A–D 转换器由 *N* 位逐次逼近寄存器 SAR、D–A 转换器、比较器、控制逻辑电路组成，如图 18–2 所示。

逐次逼近式 A–D 转换器的原理可以概括为对分搜索。例如，若模拟电压为 5V（为了说明方便，此处均为整数量），A–D 转换器为 3 位，当启动信号作用后，时钟信号在控制逻辑作用下：

1）首先使寄存器的最高位 D_2=1，其余为 0，此数字量 100B 经 D–A 转换器转换成模拟电压，即 U_o=4V，送到比较器输入端与被转换的模拟量 U_{in}=5V 进行比较，控制逻辑根据比较器的输出进行判断，因 $U_{in} \geqslant U_o$，则保留 D_2=1。

2）其次对下一位 D_1 进行比较，同样先使 D_1=1，与上一位 D_2 位一起即 110 进入 D–A 转换器，转换为 U_o=6 再进入比较器，与 U_{in}=5V 进行比较，因 $U_{in}<U_o$，则使 D_2=0。

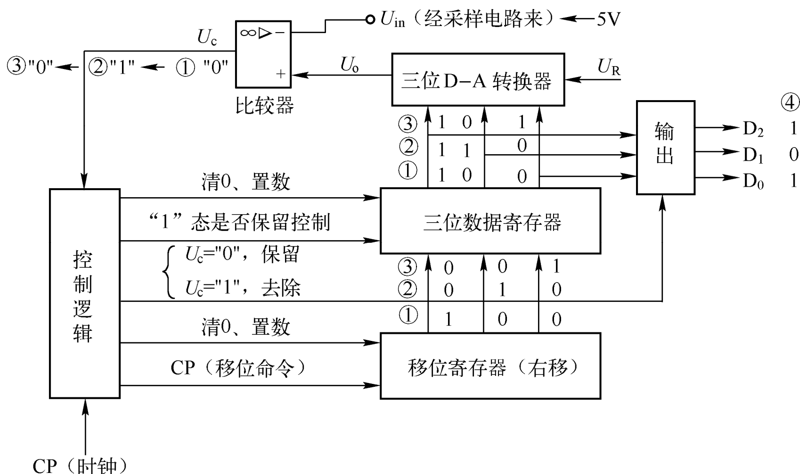


图 18–2　逐次逼近式 A–D 转换原理图

3）最后一位 D_0=1 即 101 经 D–A 转换为 U_o=5V，再与 U_{in}=5V 进行比较，因 $U_{in} \geqslant U_o$，保留 D_0=1。比较完毕，寄存器中的数字量 101 即为模拟量 5 的转换结果，存在输出锁存器中等待输出。

上述过程非常类似于用砝码称量物体的过程，见表 18–1。

表 18-1　砝码称量物体的过程

顺　序	砝码重量	比较判断	砝码去留	结果表示
1	8g	8g<13g	留	1
2	8g+4g	12g<13g	留	1
3	8g+4g+2g	14g<13g	去	0
4	8g+4g+1g	13g=13g	留	1

逐次逼近式 A-D 转换器的优点是转换速度快，转换时间固定；缺点是抗干扰能力差。

基于此原理的 A-D 转换器的常用品种有普通型 8 位单路 ADC0801～ADC0805、8 位 8 路 ADC0808/0809、8 位 16 路 ADC0816/0817 等，混合集成高速型 12 位单路 AD574A、ADC803 等。

▷▷▷ 18.4.2　A-D 转换器选择原则

A-D 转换是前向通道中的一个环节，并不是所有前向通道中都必须配备 A-D 转换器。只有模拟量输入通道，才用到 A-D 转换器。因此，首先要确定前向通道结构方案。当确定使用 A-D 转换器以后，按下列原则选择 A-D 转换器芯片：

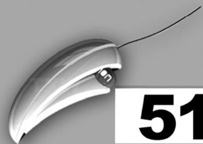
1）根据前向通道的总误差选择 A-D 转换器精度及分辨率。用户提出的数据采集精度要求是综合精度要求，包括传感器精度、信号调节电路精度和 A-D 转换精度。应将综合精度在各个环节上进行分配，以确定对 A-D 转换器的精度要求，据此确定 A-D 转换器的位数。

2）根据信号对象的变化率及转换精度要求确定 A-D 转换速度。对于快速信号要确定是否需要加采样/保持电路。因为对快速信号采集时，常常要求有很快的转换速度，这大大增加了 A-D 转换器的成本，而且有时找不到高速的 A-D 转换芯片，故对快速信号必须考虑采样/保持电路。

3）根据环境条件选择 A-D 转换芯片的一些环境参数要求，如工作温度、功耗、可靠性等级等。

4）根据计算机接口特征选择 A-D 转换器的输出状态。例如，A-D 转换器是并行输出还是串行输出；是二进制码还是 BCD 码输出；是用外部时钟、内部时钟还是不用时钟；有无转换结束状态信号；与 TTL、CMOS 及 ECL 电路的兼容性；与微机接口是否易连接等输出功能。

5）其他因素。还要考虑到成本、资源、是否是流行芯片等因素。



案例19



温度监控器

19.1 案例任务

将 DS18B20 检测的温度值在 LED 上显示出来。

19.2 案例要点

1）DS18B20 的基本特点。
2）DS18B20 的内部结构及引脚。
3）DS18B20 中温度与数字量的关系。
4）DS18B20 的单总线协议内容，重点为引脚时序。
5）DS18B20 与 51 单片机的接口电路，即驱动程序编写方法。

19.3 案例设计

19.3.1 硬件电路

温度监控器电路如图 19-1 所示。

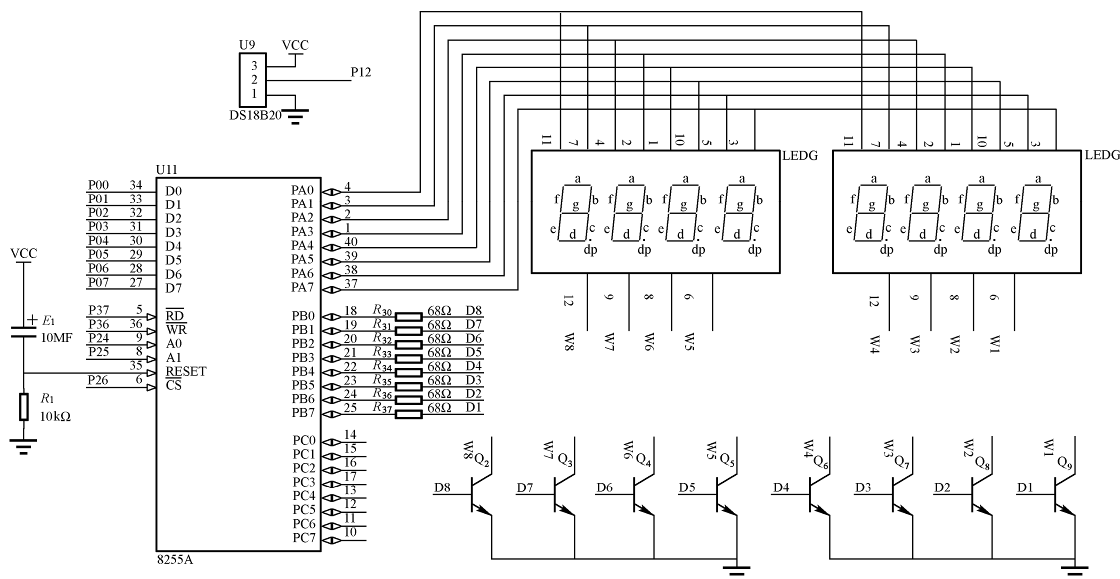


图 19-1　温度监控器电路

19.3.2 软件代码

（1）汇编程序

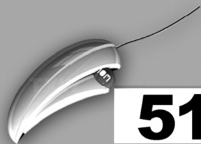


```
PORTA    EQU    8FFFH          ;8255A 口地址
PORTB    EQU    9FFFH          ;8255B 口地址
PORTC    EQU    0AFFFH         ;8255C 口地址
CADDR    EQU    0BFFFH         ;8255 控制字地址
SHOW1    EQU    70H            ;数码管 1 位数存放内存位置
SHOW2    EQU    71H            ;数码管 2 位数存放内存位置
SHOW3    EQU    72H            ;数码管 3 位数存放内存位置
SHOW4    EQU    73H            ;数码管 4 位数存放内存位置
SHOW5    EQU    74H            ;数码管 5 位数存放内存位置
SHOW6    EQU    75H            ;数码管 6 位数存放内存位置
SHOW7    EQU    76H            ;数码管 7 位数存放内存位置
SHOW8    EQU    77H            ;数码管 8 位数存放内存位置
TEM_H    EQU    78H            ;放读取的高 8 位
TEM_L    EQU    79H            ;放读取的低 8 位
DQ       BIT    P1.2           ;通信数据引脚
         ORG    0000H
         AJMP   INI
         ORG    0100H
INI:     MOV    A,#81H         ;设置 8255 工作方式
         MOV    DPTR,#CADDR
         MOVX   @DPTR,A
         MOV    DPTR,#PORTC
         MOV    A,#00H
         MOVX   @DPTR ,A
         MOV    SHOW1,#00H     ;设置数码管显示
         MOV    SHOW2,#00H
         MOV    SHOW3,#00H
         MOV    SHOW4,#00H
         MOV    SHOW5,#00H
         MOV    SHOW6,#00H
         MOV    SHOW7,#00H
         MOV    SHOW8,#00H
MAIN:    ACALL  GETTEM
         MOV    A,#0FH
         ANL    A,TEM_L        ;获得小数部分(4 位)
         MOV    B,#10
         MUL    AB
         MOV    B,#16
         DIV    AB
         MOV    DPTR,#NUMLAB
         MOVC   A,@A+DPTR
         MOV    SHOW1,A
         MOV    A,TEM_L
         SWAP   A
         MOV    TEM_L,A
         MOV    A,TEM_H
         SWAP   A
         MOV    R0,#TEM_L
         XCHD   A, @R0
HUND:    MOV    B,#100         ;百位计算
         DIV    AB
```

```
                MOV     DPTR,#NUMLAB
                MOVC    A,@A+DPTR       ;查百位数的 7 段代码
                MOV     SHOW4,A
TEN:            MOV     A,B             ;十位计算
                MOV     B,#10
                DIV     AB
                MOV     DPTR,#NUMLAB
                MOVC    A,@A+DPTR       ;查十位数的 7 段代码
                MOV     SHOW3,A
SING:           MOV     A,B             ;个位计算
                MOV     DPTR,#NUMLAB
                MOVC    A,@A+DPTR       ;查个位数的 7 段代码
                ORL     A,#80H
                MOV     SHOW2,A
RETURN:         ACALL   PLAY
                JMP     MAIN
/**********************************************/
/*            DS18B20 初始化                  */
/**********************************************/
INI1820:        SETB    DQ
                NOP
                CLR     DQ
                MOV     R2,#250
L1:             DJNZ    R2,L1
                SETB    DQ
                MOV     R2,#25
L15:            DJNZ    R2,L15
                CLR     C
                ORL     C,DQ
                JC      INI1820
                MOV     R6,#23
L16:            ORL     C,DQ
                JC      L3
                DJNZ    R6,L16
                SJMP    INI1820
L3:             MOV     R2,#120
                DJNZ    R2,$
                RET
/**********************************************/
/*            读转换后的温度值                */
/**********************************************/
GETTEM:         SETB    DQ
                LCALL   INI1820
                JB      DQ,TSS2
                RET                         ;若不存在则返回
TSS2:           MOV     A,#0CCH             ;跳过 ROM
                LCALL   WR_1820
                MOV     A,#44H              ;发出温度转换命令
                LCALL   WR_1820
                LCALL   D1MS                ;延时
                LCALL   INI1820
                MOV     A,#0CCH             ;跳过 ROM
                LCALL   WR_1820
                MOV     A,#0BEH             ;发出读温度转换命令
                LCALL   WR_1820
                LCALL   RED_1820            ;读两个字节的温度
                RET
```



```
/*********************************************/
/*              写命令到 DS18B20                */
/* 说明:        向 DS18B20 写入控制字             */
/*********************************************/
WR_1820:        CLR     EA
                MOV     R2,#8
L9:             SETB    DQ
                MOV     R3,#7
                RRC     A
                CLR     DQ
                DJNZ    R3,$
                MOV     DQ,C
                MOV     R3,#20
                DJNZ    R3,$
                DJNZ    R2,L9
                SETB    DQ
                SETB    EA
                RET
/*********************************************/
/*              读出两个字节温度数据             */
/*********************************************/
RED_1820:       MOV     R2,#2
                MOV     R1,#TEM_L        ;低位存入(TEM_L)，高位存
                                         入(TEM_H)
L7:             MOV     R3,#8
L6:             CLR     C
                CLR     DQ
                NOP
                NOP
                NOP
                SETB    DQ               ;开始读数据，总线释放
                MOV     R4,#4
                DJNZ    R4,$             ;总线保持 8μs
                MOV     C,DQ
                RRC     A
                MOV     R5,#30
L5:             DJNZ    R5,L5            ;等待 60μs 释放总线
                DJNZ    R3,L6
                MOV     @R1,A
                DEC     R1               ;指向高 8 位存储单元
                DJNZ    R2,L7
                SETB    DQ
                RET
/*********************************************/
/*              显示子程序                      */
/*********************************************/
PLAY:           MOV     R0,#08H
                MOV     R1,#SHOW1        ;取显示码
                MOV     R2,#80H
DPLOP:          MOV     A,@R1
                MOV     DPTR,#PORTA
                MOVX    @DPTR,A          ;送出个位的 7 段代码
                MOV     DPTR,#PORTB
                MOV     A,R2
                MOVX    @DPTR ,A         ;开相应的位显示
                ACALL   D1MS             ;显示 162μs
                MOV     DPTR,#PORTB
```




```
                MOV     A,#0FFH
                MOVX    @DPTR ,A        ;关闭十位显示，防止鬼影
                MOV     A,R2
                RR      A
                MOV     R2,A
                INC     R1
                DJNZ    R0,DPLOP        ;循环执行 8 次
                RET
D1MS:           MOV     R6,#150
                DJNZ    R6,$
                RET
;实验板上的 7 段数码管 0～9 数字的共阴显示代码
NUMLAB: DB      3FH,06H,5BH,4FH,66H,6DH,7DH,07H,7FH,6FH
                END
```

（2）C 语言程序

```
#include<reg51.h>
sbit        DQ=P1^2;
sbit        P1_3=0x93;
#define         TRUE                1
#define         FALSE               0

#define     PORTA           0x8FFF          //8255A 口地址
#define     PORTB           0x9FFF          //8255B 口地址
#define     PORTC           0xAFFF          //8255C 口地址
#define     CADDR           0xBFFF          //8255 控制字地址
#define     uchar           unsigned char
#define     unint           unsigned int
void        delay(unsigned int N);
uchar       Reset(void);
void        write_bit(unsigned char bitval);
void        write_Byte(unsigned char val);
uchar       read_Byte(void);
void        WriteData(unsigned int Addr,unsigned char Data);
uchar       ReadData(unsigned int Addr);
void        Testtemperature(void);
void        Printer(void);
void        Delay(void);
void        Ini(void);

unint       show[8];
code uchar  word[10]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};
void main(void)
{
        Ini();
        while(1)
        {
                Testtemperature();
                Printer();
```



```
        }
}
/*********************************************/
/*                  初始化程序                   */
/* 说明:    配置寄存器及初始化参数               */
/*********************************************/
void Ini(void)
{
      //8255 工作模式
       WriteData(CADDR,0x81);
       WriteData(PORTC,0xff);
}
/*********************************************/
/*                    采样子程序                  */
/* 说明:                  无                      */
/*********************************************/
void Testtemperature(void)
{
      uchar Hbyte,Lbyte,task;

      while(Reset());
      write_Byte(0xcc);
      write_Byte(0x44);
      while(Reset());
      write_Byte(0xcc);
      write_Byte(0xbe);
      Lbyte=read_Byte();
      Hbyte=read_Byte();
      show[0]=0x00;
      show[1]=0x00;
      show[2]=0x00;
      task=((Lbyte&0xf0)>>4)|((Hbyte&0x0f)<<4);
      show[4]=word[(task/100)];
      task=task%100;                         //取余数
      show[5]=word[task/10];
      task=task%10;
      show[6]=word[task]|0x80;
      show[7]=word[((Lbyte&0x0f)*10)/16];          //查表得小数位的值
}
/*********************************************/
/*                    延时子程序                  */
/* 说明:    实现(16*N+24)μs 的延时,              */
/*           采用 11.0592MHz 的时钟               */
/*********************************************/
void delay(unsigned int N)
{
```

```
        int i;
         for(i=0;i<N;i++);
}
/*********************************************/
/*                    复位子程序                    */
/* 说明:                  无                        */
/*********************************************/
unsigned char Reset(void)
{
        uchar deceive_ready;
         DQ=1;
         delay(3);
         DQ=0;
         delay(29);
         DQ=1;
         delay(3);
         deceive_ready=DQ;
         delay(25);
         return(deceive_ready);
}
/*********************************************/
/*                  读 1bit 子程序                   */
/* 说明:        返回接收的数据                       */
/*********************************************/
uchar read_bit(void)
{
        uchar i;
         DQ=0;
         DQ=1;
         for(i=0;i<3;i++);
         return(DQ);
}
/*********************************************/
/*                  写 1bit 子程序                   */
/* 说明:                 无                         */
/*********************************************/
void write_bit(uchar bitval)
{
        DQ=0;
         if(bitval==1)
         DQ=1;
         delay(5);
         DQ=1;
}
/*********************************************/
/*                  写 1byte 子程序                  */
```



```
/* 说明:              无                              */
/*********************************************/
void write_Byte(uchar val)
{
      uchar i,temp;
       EA=0;
       for(i=0;i<8;i++)
       {
             temp=val>>i;
              temp=temp&0x01;
              write_bit(temp);
              delay(5);
       }
       EA=1;
}
/*********************************************/
/*                  读 1byte 子程序                  */
/* 说明:        返回接收的数据 value                */
/*********************************************/
uchar read_Byte(void)
{
      uchar i,m,receive_data;
       m=1;
       receive_data=0;
       EA=0;
       for(i=0;i<8;i++)
       {
             if(read_bit())
              {
                    receive_data=receive_data+(m<<i);
              }
              delay(6);
       }
       EA=1;
       return(receive_data);
}
/*********************************************/
/*                  显示子程序                      */
/* 说明:              无                              */
/*********************************************/
void Printer(void)
{
       unint i;
       uchar MASK,mask=0x01;
       for(i=0;i<8;i++)
       {
```

```
            MASK=mask;
            mask=mask<<1;
            WriteData(PORTA,show[i]);
            WriteData(PORTB,MASK);
            Delay();
            WriteData(PORTA,0x00);
            WriteData(PORTB,MASK);
        }
}
/*******************************************/
/*              写外部寄存器                */
/* 说明:            无                      */
/*******************************************/
void WriteData(unint Addr,uchar Data)
{
    *((uchar xdata *)Addr)=Data;
}
/*******************************************/
/*              读外部寄存器                */
/* 说明:            无                      */
/*******************************************/
unsigned char ReadData(unsigned int Addr)
{
    return *((unsigned char xdata *)Addr);
}
/*******************************************/
/*                延时程序                  */
/* 说明:             无                     */
/*******************************************/
void Delay(void)
{
    int i,j;
     for(i=0;i<20;i++)
         for(j=0;j<5;j++);
}
```

▷▷ 19.4 案例笔记

▷▷▷ 19.4.1 单总线

1-Wire 单总线是 Maxim 全资子公司 Dallas 的一项专有技术。与目前多数标准串行数据通信方式，如 SPI/I^{2}C/MICROWIRE 不同，它采用单根信号线，既传输时钟，又传输数据，而且数据传输是双向的。具有节省 I/O 口线资源、结构简单、成本低廉、便于总线扩展和维护等诸多优点。

1-Wire 总线由一个总线主节点、一个或多个从节点组成系统，通过一根信号线对从芯片进行数据的读取。每一个符合 1-Wire 协议的从芯片都有一个唯一的地址，包括 48 位的序列号、8 位的家族代码和 8 位的 CRC 代码。主芯片对各个从芯片的寻址依据这 64 位的不同来进行。1-Wire 总线利用一根线实现双向通信，因此其协议对时序的要求较严格，如应答等时序都有明确的时间要求。基本的时序包括复位及应答时序、写一位时序、读一位时序。在复位及应答时序中，主器件发出复位信号后，要求从器件在规定的时间内送回应答信号；在位读和位写时序中，主器件要在规定的时间内读回或写出数据。1-Wire 单总线适用于单个主机系统，能够控制一台或多台从机设备。主机可以是微控制器，从机可以是单总线器件，它们之间的数据交换只通过一条信号线。当只有一台从机位于总线上时系统可按照单节点系统操作；而当多台从机位于总线上时，则系统按照多节点系统操作。

作为一种单主机多从机的总线系统，在一条 1-Wire 总线上可挂接的从器件数量几乎不受限制。为了不引起逻辑上的冲突，所有从器件的 1-Wire 总线接口都是漏极开路的，因此在使用时必须对总线外加上拉电阻（一般取 5kΩ 左右）。主机对 1-Wire 总线的基本操作分为复位、读和写三种，其中所有的读/写操作均为低位在前、高位在后。复位、读和写是 1-Wire 总线通信的基础，下面通过具体程序详细介绍这三种操作的时序要求（程序中 DQ 代表 1-Wire 总线，定义为 P1.0，uchar 定义为 unsigned char）。

（1）1-Wire 总线的复位

复位是 1-Wire 总线通信中最为重要的一种操作，在每次总线通信之前主机必须首先发送复位信号。产生复位信号时主机首先将总线拉低 480～960μs 然后释放，由于上拉电阻的存在，此时总线变为高电平。1-Wire 总线器件在接收到有效跳变的 15～60μs 内会将总线拉低 60～240μs，在此期间内主机可以通过对 DQ 采样来判断是否有从器件挂接在当前总线上。函数 Reset( )的返回值为 0 表示有器件挂接在总线上，返回值为 1 则表示没有器件挂接在总线上。总线复位程序代码如下：

```
uchar Reset(void)
{
 uchar tdq;
 DQ=0;            //主机拉低总线
 delay480μs();    //等待 480μs
 DQ=1;            //主机释放总线
 delay60μs();     //等待 60μs
 tdq=DQ;          //主机对总线采样
 delay480μs();    //等待复位结束
 return tdq;      //返回采样值
}
```

（2）1-Wire 总线的写操作

由于只有一条 I/O 线，主机 1-Wire 总线的写操作只能逐位进行，连续写 8 次即可写入总线一个字节。向总线写 1bit 程序代码如下：

```
void Writebit(uchar wbit)
{
```

```
 _nop_();
 //保证两次写操作间隔 1μs 以上
 DQ=0;
 _nop_();
 //保证主机拉低总线 1μs 以上
 if(wbit)
 {

//向总线写 1
 DQ=1;
 delay60μs();
 }

else
 {

//向总线写 0
 delay60μs();
 DQ=0;
 }
}
```

当 51 单片机的时钟频率为 12MHz 时，程序中的语句_nop_();可以产生 1μs 的延时，调用此函数时需包含头文件“intrins.h”。向 1-Wire 总线写 1bit 至少需要 60μs，同时还要保证两次连续的写操作有 1μs 以上的间隔。若待写位为 0 则主机拉低总线 60μs 然后释放，写 0 操作完成。若待写位为 1，则主机拉低总线并在 1～15μs 内释放，然后等待 60μs，写 1 操作完成。

（3）1-Wire 总线的读操作

与写操作类似，主机对 1-Wire 总线的读操作也只能逐位进行，连续读 8 次，即可读入主机一个字节。从 1-Wire 总线读取 1bit 同样至少需要 60μs，同时也要保证两次连续的读操作间隔 1μs 以上。从总线读 1bit 程序代码如下：

```
uchar Readbit()
{
 uchar tdq;
 _nop_();
 //保证两次连续写操作间隔 1μs 以上
 DQ=0;

_nop_();

//保证拉低总线的时间不少于 1μs

DQ=1;
 _nop_();
```

```
  tdq=DQ;
//主机对总线采样
delay60μs();
//等待读操作结束
  return tdq;
//返回读取到的数据
}
```

从总线读数据时，主机首先拉低总线 1μs 以上然后释放，在释放总线后的 1～15μs 内主机对总线的采样值即为读取到的数据。

▷▷▷ 19.4.2　DS18B20 总结

（1）DS18B20 基本特点

1）支持“单总线”接口的温度传感器，仅需要一个端口引脚就可以与单片机进行通信，无需其他外围元器件。

2）在 DS18B20 中的每个器件上都有独一无二的序列号，因此多个 DS18B20 可以并联在唯一的三线上，实现多点组网功能。

3）实际应用中不需要外部任何元器件即可实现测温。

4）测量温度范围为−55～+125℃，在−10～+85℃范围内，精度为±0.5℃。

5）数字温度计的分辨率用户可以 9～12 位选择，精度为±0.5℃。

6）内部有温度上、下限告警设置。设定的报警温度存储在 E^2PROM 中，掉电后依然保存。

7）可通过数据线供电（寄生供电），电压范围为 3.0～5.5V。也可以采用外部 5V 电源供电。

8）负电压特性，电源极性接反时，温度计不会烧毁，但不能正常工作。

9）温度转换时间最长为 750ms。

（2）DS18S20 的内部结构及引脚

DS18B20 内部结构如图 19-2 所示，主要由四部分组成：64 位光刻 ROM、温度传感器、非挥发的温度报警触发器 TH 和 TL、配置寄存器。

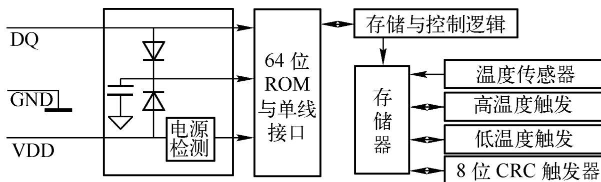


图 19-2　DS18B20 内部结构

内部结构中，光刻 ROM 的 64 位序列号是出厂前被光刻好的，它可以看作是该 DS18B20 的地址序列码。64 位光刻 ROM 的排列是：开始 8 位（28H）是产品类型标号，接

着的 48 位是该 DS18B20 自身的序列号，最后 8 位是前面 56 位的循环冗余校验码（CRC=X8+X5+X4+1）。光刻 ROM 的作用是使每一个 DS18B20 都各不相同，这样就可以实现一根总线上挂接多个 DS18B20 的目的。

DS18S20 采用 3 引脚 PR35 封装（或 8 引脚 SOIC 封装），如图 19-3 所示，各引脚功能见表 19-1。

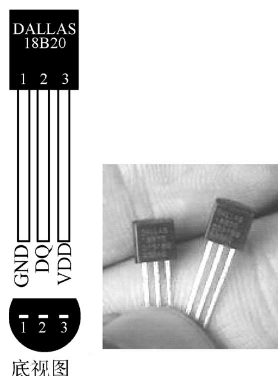


图 19-3　DS18B20 引脚图

表 19-1　DS18B20 详细引脚功能描述

名　　称	引脚功能描述
GND	地信号
DQ	数据输入/输出引脚。开漏单总线接口引脚。当被用在寄生电源下，也可以向器件提供电源
VDD	可选择的 VDD 引脚。当工作于寄生电源时，此引脚必须接地

（3）温度与数字量的关系

DS18B20 中的温度传感器可完成对温度的测量，以 12 位转化为例，用 16 位符号扩展的二进制补码读数形式提供，以 0.0625℃/LSB 形式表达，见表 19-2，其中 S 为符号位。

表 19-2　温度与数字量的关系

	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
LS byte	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
MS byte	S	S	S	S	S	2^6	2^5	2^4

这是 12 位转化后得到的 12 位数据，存储在 DS18B20 的两个 8bit RAM 中，二进制中的前面 5 位是符号位，如果测得的温度大于 0，这 5 位为 0，只要将测到的数值乘以 0.0625 即可得到实际温度；如果温度小于 0，这 5 位为 1，测到的数值需要取反加 1 再乘以 0.0625 即可得到实际温度。

例如，+125℃的数字输出为 07D0H，+25.0625℃的数字输出为 0191H，−25.0625℃的数字输出为 FF6FH，−55℃的数字输出为 FC90H。

程序中假设只处理 0℃以上的温度（即温度值都为正），因此 Testtemperature(void)子程序中，task=((Lbyte&0xf0)>>4)|((Hbyte&0x0f)<<4)处理了温度采样后整数的结果，即高字节的低 4 位与低字节的高 4 位构成了采样数据的整数部分，而低字节的低 4 位为小数，在 show[7]=word[((Lbyte&0x0f)*10)/16];语句中处理，除 16 相当于乘 0.0625，乘 10 的目的是取整显示。

（4）DS18B20 使用中的注意事项

DS18B20 虽然具有测温系统简单、测温精度高、连接方便、占用口线少等优点，但在实际应用中也应注意以下几方面的问题：

1）较小的硬件开销需要相对复杂的软件编程进行补偿，由于 DS18B20 与微处理器间采用串行数据传送，因此，在对 DS18B20 进行读/写编程时，必须严格地保证读写时序，否则将无法读取测温结果。在使用 PL/M、C 等高级语言进行系统程序设计时，对 DS18B20 操作部分最好采用汇编语言实现。

2）在 DS18B20 的有关资料中均未提及单总线上所挂 DS18B20 数量问题，容易使人误认为可以挂任意多个 DS18B20，在实际应用中并非如此。当单总线上所挂 DS18B20 超过 8 个时，就需要解决微处理器的总线驱动问题，这一点在进行多点测温系统设计时要加以注意。

3）连接 DS18B20 的总线电缆是有长度限制的。实验中，当采用普通信号电缆传输长度超过 50m 时，读取的测温数据将发生错误。当将总线电缆改为双绞线带屏蔽电缆时，正常通信距离可达 150m，当采用每米绞合次数更多的双绞线带屏蔽电缆时，正常通信距离进一步加长。这种情况主要是由总线分布电容使信号波形产生畸变造成的。因此，在用 DS18B20 进行长距离测温系统设计时要充分考虑总线分布电容和阻抗匹配问题。

4）在 DS18B20 测温程序设计中，向 DS18B20 发出温度转换命令后，程序总要等待 DS18B20 的返回信号，一旦某个 DS18B20 接触不好或断线，当程序读 DS18B20 时，将没有返回信号，程序进入死循环。这一点在进行 DS18B20 硬件连接和软件设计时也要给予一定的重视。

5）由于 DS18B20 采用的是 1-Wire 总线协议方式，即在一根数据线上实现数据的双向传输，而对 51 单片机来说，硬件上并不支持单总线协议，因此，必须采用软件的方法来模拟单总线的协议时序，从而完成对 DS18B20 芯片的访问。

6）由于 DS18B20 是在一根 I/O 线上读写数据，因此，对读写的数据位有着严格的时序要求。DS18B20 有严格的通信协议来保证各位数据传输的正确性和完整性。该协议定义了几种信号的时序：初始化时序、读时序、写时序。所有时序都是将主机作为主设备，单总线器件作为从设备。而每一次命令和数据的传输都是从主机主动启动写时序开始，如果要求单总线器件回送数据，在进行写命令后，主机需启动读时序完成数据接收。数据和命令的传输都是低位在先。

案例20

 数字电子钟

20.1 案例任务

上电启动后，系统会首先进入日期显示模式，此时，数码管上会显示年、月、日和星期。操作者可以按矩阵键盘上的 A 键切换到时间显示模式，时间模式会显示当前的时、分、秒信息。操作者按下矩阵键盘上的 C 键切换到定时模式，可以对定时器的时、分进行设置。在上述三种模式下，按下矩阵键盘上的 B 键就可以进入设置状态。在时间模式下，如果检测到达到预设时间，单片机就会发出铃声，实现定时功能。若要取消定时，转为定时界面按下矩阵键盘上的 D 键。按键功能总结如下：

1）0～9：输入数字。

2）A：日期、时分秒显示切换。

3）B：修改日期、时分秒和定时时间。

4）C：定时时间显示。

5）D：定时使能切换。

20.2 案例要点

1）DS1302 的基本特点。

2）DS1302 的内部结构及引脚和功能。

3）DS1302 的 SPI 时序。

4）DS1302 与 51 单片机的硬件连接。

5）DS1302 的初始化程序及驱动程序编写。

20.3 案例设计

20.3.1 硬件电路

数字电子钟电路如图 20-1 所示。

20.3.2 软件代码

1. 主程序流程图

该系统主要由两个功能组成，即时间和日期的显示功能以及定时闹铃的功能，与此相对，系统软件也由时间和日期显示软件及定时闹铃软件构成。

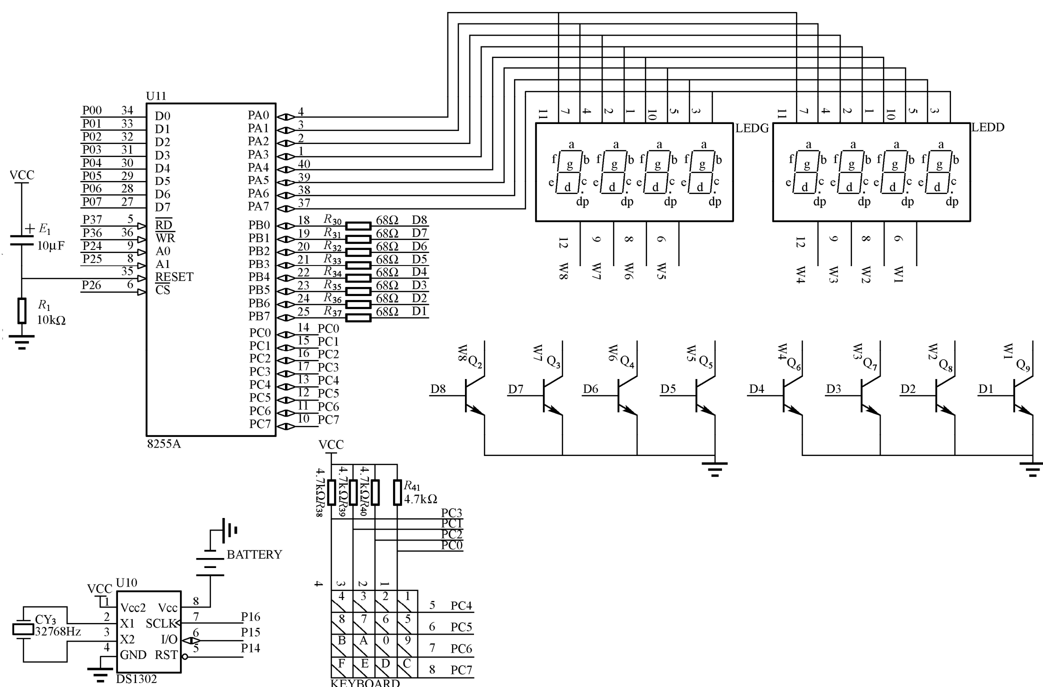


图 20-1　数字电子钟电路

数字电子钟程序流程图如图 20-2 所示。

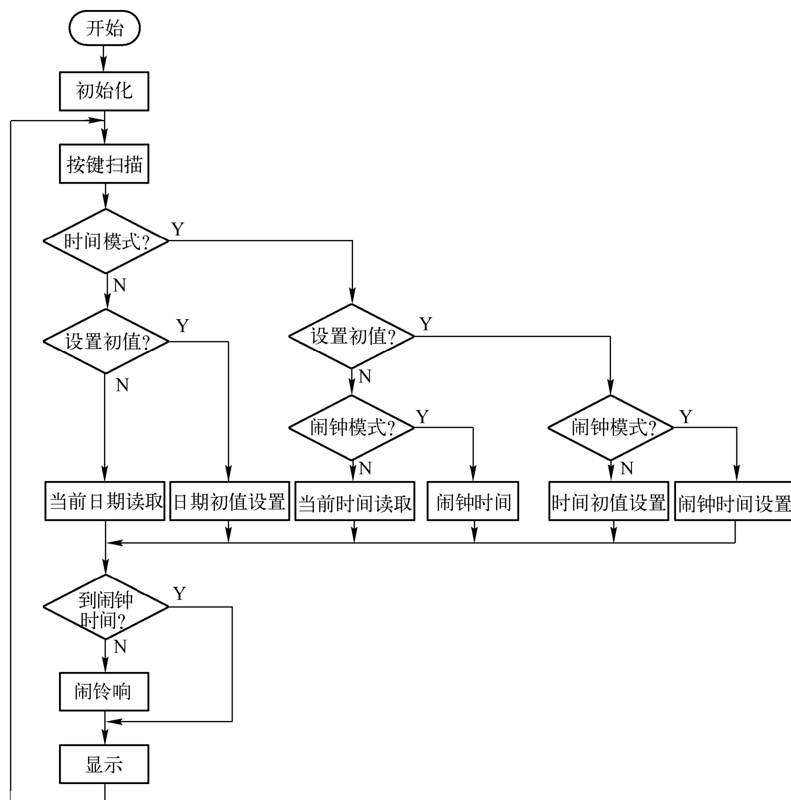


图 20-2　数字电子钟程序流程图

2. 定时闹铃子程序分析

该段功能的实现时通过将定时寄存器中的数据（时、分、秒）与当前的时间数据进行比较，如果相等则将闹铃开标志位置 1，系统会自动监测该标志位，并及时开启闹钟。

C 语言代码如下：

```
if(clkenable&&(ReadHr()==clk_hr)&&(ReadMin()==clk_min))
      clkflag=1;
else
      clkflag=0;
```

中断处理函数中，查询到开启闹钟的标志位为 1 后就会进行相应的处理，使蜂鸣器发声。在定时器中断处理函数中加入蜂鸣器发声的处理，既可以实现报警功能，又不影响单片机对数码管及其他元器件的操作，达到了实时的效果。

3. 数字电子钟源代码

（1）汇编程序

```
PORTA       EQU     8FFFH       ;8255A 口地址
PORTB       EQU     9FFFH       ;8255B 口地址
PORTC       EQU     0AFFFH      ;8255C 口地址
CADDR       EQU     0BFFFH      ;8255 控制字地址
ADC08       EQU     0F0FFH
TIMEH_C     EQU     0FAH        ;定时器初值
TIMEL_C     EQU     00H         ;定时器初值
TIMEMOD     EQU     15H         ;定时器工作模式
SHOW1       EQU     20H         ;数码管 1 位数存放内存位置
SHOW2       EQU     21H         ;数码管 2 位数存放内存位置
SHOW3       EQU     22H         ;数码管 3 位数存放内存位置
SHOW4       EQU     23H         ;数码管 4 位数存放内存位置
SHOW5       EQU     24H         ;数码管 5 位数存放内存位置
SHOW6       EQU     25H         ;数码管 6 位数存放内存位置
SHOW7       EQU     26H         ;数码管 7 位数存放内存位置
SHOW8       EQU     27H         ;数码管 8 位数存放内存位置
KEYNUM      EQU     30H         ;键盘码
KEYCNT      EQU     31H         ;键盘计数器
T_CLK       BIT     P1.6        ;实时时钟时钟线引脚
T_IO        BIT     P1.5        ;实时时钟数据线引脚
T_RST       BIT     P1.4        ;实时时钟复位线引脚
KEYPRES     BIT     41H         ;按键监视位
MODE        BIT     70H         ;工作模式位
CLKFLAG     BIT     71H         ;定时闹钟模式位
FLASH       BIT     72H         ;闪烁位
RING        BIT     73H         ;蜂鸣器使能位
WORN        BIT     74H         ;闹铃响使能位
CLKEN       BIT     75H         ;闹钟使能位
SETFLAG     BIT     77H         ;设置监控位
SEC_S       BIT     60H         ;设置秒
MIN_S       BIT     61H         ;设置分
HOR_S       BIT     62H         ;设置时
YEA_S       BIT     63H         ;设置年
```



```
MON_S     BIT     64H                 ;设置月
DAY_S     BIT     65H                 ;设置日
WEE_S     BIT     66H                 ;设置星期
SECOND    EQU     50H                 ;秒寄存器
MINUTE    EQU     51H                 ;分寄存器
HOUR      EQU     52H                 ;时寄存器
DAY       EQU     53H                 ;日寄存器
MONTH     EQU     54H                 ;月寄存器
WEEK      EQU     55H                 ;星期寄存器
YEAR      EQU     56H                 ;年寄存器
SECCLK    EQU     60H                 ;秒寄存器
MINCLK    EQU     61H                 ;分寄存器
HORCLK    EQU     62H                 ;时寄存器
          ORG     0000H
          AJMP    INI
          ORG     001BH
          AJMP    TIME1               ;定时器 1 中断入口
          ORG     0100H
INI:      MOV     A,#81H              ;设置 8255 工作方式
          MOV     DPTR,#CADDR
          MOVX    @DPTR,A
          MOV     DPTR,#PORTC
          MOV     A,#00H
          MOVX    @DPTR ,A
          MOV     TMOD,#TIMEMOD       ;设置定时器
          MOV     TH1,#TIMEH_C
          MOV     TL1,#TIMEL_C
          SETB    TR1
          SETB    ET1
          SETB    EA
          MOV     SHOW1,#00H          ;初始化数码管显示内容
          MOV     SHOW2,#00H
          MOV     SHOW3,#00H
          MOV     SHOW4,#00H
          MOV     SHOW5,#00H
          MOV     SHOW6,#00H
          MOV     SHOW7,#00H
          MOV     SHOW8,#00H
          MOV     SECCLK,#00H         ;初始化闹铃时间
          MOV     MINCLK,#45H
          MOV     HORCLK,#23H
          MOV     10H,#240
          CLR     SEC_S               ;初始化标志位
          CLR     MIN_S
          CLR     HOR_S
          CLR     DAY_S
          CLR     MON_S
          CLR     YEA_S
          CLR     WEE_S
          SETB    MODE
          CLR     FLASH
          CLR     RING
          CLR     WORN
          CLR     CLKEN
    ;************主程序**************************
MAIN:     MOV     SECOND,#55H         ;赋初值
```

```
                MOV     MINUTE,#58H
                MOV     HOUR,#22H
                MOV     DAY,#18H
                MOV     MONTH,#05H
                MOV     WEEK,#05H
                MOV     YEAR,#07H
                LCALL   SET1302
CLOCK:          MOV     KEYNUM,#00H             ;按键扫描
                ACALL   SCAN
                MOV     A,KEYNUM
SETPUP:         CJNE    A,#0AH,SETUP1           ;按键 A 切换时间/日期工作方式
                CPL     MODE
                CLR     KEYPRES
SETUP1:         CJNE    A,#0BH,SETUP2           ;按键 B 设置时间/日期及闹钟初值
                CPL     SETFLAG
                MOV     KEYCNT,#00H
                CLR     KEYPRES
                CLR     SEC_S
                CLR     MIN_S
                CLR     HOR_S
                CLR     DAY_S
                CLR     MON_S
                CLR     YEA_S
                CLR     WEE_S
SETUP2:         CJNE    A,#0CH,SETUP3           ;按键 C 切换时间/闹钟工作方式
                CPL     CLKFLAG
                CLR     SETFLAG
                CLR     KEYPRES
SETUP3:         CJNE    A,#0DH,GET              ;按键 D，闹铃禁止/使能
                CPL     CLKEN
                CLR     SETFLAG
                CLR     KEYPRES
GET:            CLR     EA                      ;读取当前时间及日期值
                LCALL   GET1302
                SETB    EA
                MOV     A,HOUR                  ;判断是否到定时时间
                CJNE    A,HORCLK,NOWORN
                MOV     A,MINUTE
                CJNE    A,MINCLK,NOWORN
                SETB    WORN                    ;若到闹钟时间则闹铃标志位置 1
                JMP     MODCHOS
NOWORN:         CLR     WORN                    ;否则闹铃标志位置 0
MODCHOS:        MOV     C,MODE                  ;时间/日期模式选择
                JNC     TIME
                JMP     DATE
;时间工作模式
TIME:           MOV     C,SETFLAG               ;判断是否处于设置模式
                JC      SHEZHI1
                AJMP    XIANSHI1
SHEZHI1:        MOV     A,KEYCNT                ;时间设置模式
                CJNE    A,#00H,WAIT1
                CLR     SEC_S
                CLR     MIN_S
                SETB    HOR_S
WAIT1:          MOV     C,KEYPRES
                JC      HOR_H
                AJMP    XIANSHI1
```



```
HOR_H:      MOV     A,KEYCNT            ;设置小时的十位
            CJNE    A,#01H,HOR_L
            MOV     A,KEYNUM
            CLR     C
            SUBB    A,#03H              ;设置值域（十位小于 2）
            JC      PAS_H
            MOV     A,KEYCNT
            DEC     A
            MOV     KEYCNT,A
            CLR     KEYPRES
            JMP     HOR_L
PAS_H:      MOV     C,CLKFLAG
            ;输入数据有效，判断是时间设置模式，还是闹钟设置模式
            JC      CLHOR_H
            MOV     A,HOUR              ;时间设置模式
            ANL     A,#0FH
            MOV     HOUR,A
            MOV     A,KEYNUM
            SWAP    A
            ORL     A,HOUR
            MOV     HOUR,A
            LCALL   SET1302
            CLR     KEYPRES
            JMP     HOR_L
CLHOR_H:    MOV     A,HORCLK            ;闹钟设置模式
            ANL     A,#0FH
            MOV     HORCLK,A
            MOV     A,KEYNUM
            SWAP    A
            ORL     A,HORCLK
            MOV     HORCLK,A
            CLR     KEYPRES
HOR_L:      MOV     A,KEYCNT            ;小时的个位设置
            CJNE    A,#02H,MIN_H
            MOV     A,HOUR
            ANL     A,#0F0H
            CJNE    A,#20H,PAS_HL       ;十位为 2 时个位要小于 4
            MOV     A,KEYNUM
            CLR     C
            SUBB    A,#04H
            JC      PAS_HL
            MOV     A,KEYCNT
            DEC     A
            MOV     KEYCNT,A
            CLR     KEYPRES
            JMP     MIN_H
PAS_HL:     MOV     C,CLKFLAG
            ;输入数据有效，判断是时间设置模式，还是闹钟设置模式
            JC      CLHOR_L
            MOV     A,HOUR              ;时间设置模式
            ANL     A,#0F0H
            MOV     HOUR,A
            MOV     A,KEYNUM
            ORL     A,HOUR
            MOV     HOUR,A
            LCALL   SET1302
            CLR     KEYPRES
```

```
            CLR     SEC_S
            SETB    MIN_S
            CLR     HOR_S
            JMP     MIN_H
CLHOR_L:    MOV     A,HORCLK            ;闹钟设置模式
            ANL     A,#0F0H
            MOV     HORCLK,A
            MOV     A,KEYNUM
            ORL     A,HORCLK
            MOV     HORCLK,A
            CLR     KEYPRES
            CLR     SEC_S
            SETB    MIN_S
            CLR     HOR_S
MIN_H:      MOV     A,KEYCNT            ;分钟的十位设置
            CJNE    A,#03H,MIN_L
            MOV     A,KEYNUM
            CLR     C
            SUBB    A,#06H              ;设置值域（十位小于 6）
            JC      PAS_M
            MOV     A,KEYCNT
            DEC     A
            MOV     KEYCNT,A
            CLR     KEYPRES
            JMP     MIN_L
            ;输入数据有效，判断是时间设置模式，还是闹钟设置模式
PAS_M:      MOV     C,CLKFLAG
            JC      CLMIN_H
            MOV     A,MINUTE            ;时间设置模式
            ANL     A,#0FH
            MOV     MINUTE,A
            MOV     A,KEYNUM
            SWAP    A
            ORL     A,MINUTE
            MOV     MINUTE,A
            LCALL   SET1302
            CLR     KEYPRES
            JMP     MIN_L
CLMIN_H:    MOV     A,MINCLK            ;闹钟设置模式
            ANL     A,#0FH
            MOV     MINCLK,A
            MOV     A,KEYNUM
            SWAP    A
            ORL     A,MINCLK
            MOV     MINCLK,A
            CLR     KEYPRES
MIN_L:      MOV     C,CLKFLAG           ;分钟的个位设置
            JC      CLMIN_L
            MOV     A,KEYCNT            ;时间设置模式
            CJNE    A,#04H,SEC_H
            MOV     A,MINUTE
            ANL     A,#0F0H
            MOV     MINUTE,A
            MOV     A,KEYNUM
            ORL     A,MINUTE
            MOV     MINUTE,A
            LCALL   SET1302
```



```
            CLR     KEYPRES
            SETB    SEC_S
            CLR     MIN_S
            CLR     HOR_S
            JMP     SEC_H
CLMIN_L:    MOV     A,KEYCNT            ;闹钟设置模式
            CJNE    A,#04H,SEC_H
            MOV     A,MINCLK
            ANL     A,#0F0H
            MOV     MINCLK,A
            MOV     A,KEYNUM
            ORL     A,MINCLK
            MOV     MINCLK,A
            CLR     KEYPRES
            SETB    SEC_S
            CLR     MIN_S
            CLR     HOR_S
            CLR     SETFLAG
            MOV     KEYCNT,#00H
            CLR     SEC_S
            CLR     MIN_S
            CLR     HOR_S
SEC_H:      MOV     A,KEYCNT            ;秒的十位设置
            CJNE    A,#05H,SEC_L
            MOV     A,KEYNUM
            CLR     C
            SUBB    A,#06H              ;设置值域（十位小于6）
            JC      PAS_S
            MOV     A,KEYCNT
            DEC     A
            MOV     KEYCNT,A
            CLR     KEYPRES
            JMP     SEC_L
PAS_S:      MOV     A,SECOND            ;输入数据有效
            ANL     A,#0FH              ;设置时间模式
            MOV     SECOND,A
            MOV     A,KEYNUM
            SWAP    A
            ORL     A,SECOND
            MOV     SECOND,A
            LCALL   SET1302
            CLR     KEYPRES
SEC_L:      MOV     A,KEYCNT            ;秒的个位设置
            CJNE    A,#06H,XIANSHI1
            MOV     A,SECOND
            ANL     A,#0F0H
            MOV     SECOND,A
            MOV     A,KEYNUM
            ORL     A,SECOND
            MOV     SECOND,A
            LCALL   SET1302
            CLR     KEYPRES
            CLR     SETFLAG
            MOV     KEYCNT,#00H
            CLR     SEC_S
            CLR     MIN_S
            CLR     HOR_S
XIANSHI1:   MOV     C,CLKFLAG           ;时间显示模式
```

```
                        ;判断是显示闹钟的秒，还是显示当前时间的秒
          JNC     SHOWSEC
CLKSEC:   MOV     SHOW3,#00H          ;显示闹钟的秒
          MOV     SHOW2,#58H
          MOV     SHOW1,#38H
          MOV     C,CLKEN
          JC      MINMOD              ;判断是否闹钟使能
          MOV     SHOW3,#01H
          JMP     MINMOD
SHOWSEC:  MOV     B,SECOND            ;显示当前时间的秒
SECPLY:   MOV     A,B
          ANL     A,#0FH
          MOV     DPTR,#NUMLAB
          MOVC    A,@A+DPTR
          MOV     SHOW1,A
          MOV     A,B
          SWAP    A
          ANL     A,#0FH
          MOV     DPTR,#NUMLAB
          MOVC    A,@A+DPTR
          MOV     SHOW2,A
          MOV     C,SEC_S             ;设置秒时的闪烁处理
          JNC     NEXT1
          MOV     C,FLASH
          JNC     NEXT1
          MOV     SHOW1,#00H
          MOV     SHOW2,#00H
NEXT1:    MOV     SHOW3,#40H
MINMOD:   MOV     C,CLKFLAG
                        ;判断是显示当前时间的分钟，还是显示闹钟的分钟
          JNC     SHOWMIN
CLKMIN:   MOV     B,MINCLK            ;显示闹钟的分钟
          JMP     MINPLY
SHOWMIN:  MOV     B,MINUTE            ;显示当前时间的分钟
MINPLY:   MOV     A,B
          ANL     A,#0FH
          MOV     DPTR,#NUMLAB
          MOVC    A,@A+DPTR
          MOV     SHOW4,A
          MOV     A,B
          SWAP    A
          ANL     A,#0FH
          MOV     DPTR,#NUMLAB
          MOVC    A,@A+DPTR
          MOV     SHOW5,A
          MOV     C,MIN_S             ;设置该位时的闪烁处理
          JNC     NEXT2
          MOV     C,FLASH
          JNC     NEXT2
          MOV     SHOW4,#00H
          MOV     SHOW5,#00H
NEXT2:    MOV     SHOW6,#40H
HORMOD:   MOV     C,CLKFLAG
                        ;判断是显示当前时间的小时，还是显示闹钟的小时
          JNC     SHOWHOR
CLKHOR:   MOV     B,HORCLK            ;显示闹钟的小时
          JMP     HORPLY
```

```
SHOWHOR:    MOV     B,HOUR          ;显示当前时间的小时
HORPLY:     MOV     A,B
            ANL     A,#0FH
            MOV     DPTR,#NUMLAB
            MOVC    A,@A+DPTR
            MOV     SHOW7,A
            MOV     A,B
            SWAP    A
            ANL     A,#0FH
            MOV     DPTR,#NUMLAB
            MOVC    A,@A+DPTR
            MOV     SHOW8,A
            MOV     C,HOR_S         ;设置该位时的闪烁处理
            JC      ELSE1
            JMP     DISP
ELSE1:      MOV     C,FLASH
            JC      ELSE2
            JMP     DISP
ELSE2:      MOV     SHOW7,#00H
            MOV     SHOW8,#00H
            JMP     DISP
;日期工作模式
DATE:       MOV     C,SETFLAG       ;判断是否处于设置模式
            JC      SHEZHI2
            AJMP    XIANSHI2
SHEZHI2:    MOV     A,KEYCNT        ;日期设置模式
            CJNE    A,#00H,WAIT2
            CLR     WEE_S
            CLR     DAY_S
            CLR     MON_S
            SETB    YEA_S
WAIT2:      MOV     C,KEYPRES
            JC      YEA_H
            AJMP    XIANSHI2
YEA_H:      MOV     A,KEYCNT        ;设置年的十位
            CJNE    A,#01H,YEA_L
            MOV     A,YEAR
            ANL     A,#0FH
            MOV     YEAR,A
            MOV     A,KEYNUM
            SWAP    A
            ORL     A,YEAR
            MOV     YEAR,A
            LCALL   SET1302
            CLR     KEYPRES
YEA_L:      MOV     A,KEYCNT        ;设置年的个位
            CJNE    A,#02H,MON_H
            MOV     A,YEAR
            ANL     A,#0F0H
            MOV     YEAR,A
            MOV     A,KEYNUM
            ORL     A,YEAR
            MOV     YEAR,A
            LCALL   SET1302
            CLR     KEYPRES
            CLR     WEE_S
            CLR     DAY_S
            SETB    MON_S
```



```
            CLR     YEA_S
MON_H:      MOV     A,KEYCNT            ;设置月的十位
            CJNE    A,#03H,MON_L
            MOV     A,KEYNUM
            CLR     C
            SUBB    A,#02H              ;设置值域（十位小于 2）
            JC      PAS_MO
            MOV     A,KEYCNT
            DEC     A
            MOV     KEYCNT,A
            CLR     KEYPRES
            JMP     MON_L
PAS_MO:     MOV     A,MONTH             ;输入数据有效，设置月的十位
            ANL     A,#0FH
            MOV     MONTH,A
            MOV     A,KEYNUM
            SWAP    A
            ORL     A,MONTH
            MOV     MONTH,A
            LCALL   SET1302
            CLR     KEYPRES
MON_L:      MOV     A,KEYCNT            ;设置月的个位
            CJNE    A,#04H,DAY_H
            MOV     A,MONTH
            ANL     A,#0F0H
            CJNE    A,#10H,PAS_ML
            MOV     A,KEYNUM
            CLR     C
            SUBB    A,#03H              ;十位为 1 时个位小于 3
            JC      PAS_ML
            MOV     A,KEYCNT
            DEC     A
            MOV     KEYCNT,A
            CLR     KEYPRES
            JMP     DAY_H
PAS_ML:     MOV     A,MONTH             ;输入数据有效，设置月的个位
            ANL     A,#0F0H
            MOV     MONTH,A
            MOV     A,KEYNUM
            ORL     A,MONTH
            MOV     MONTH,A
            LCALL   SET1302
            CLR     KEYPRES
            CLR     WEE_S
            SETB    DAY_S
            CLR     MON_S
            CLR     YEA_S
DAY_H:      MOV     A,KEYCNT            ;设置日的十位
            CJNE    A,#05H,DAY_L
            MOV     A,KEYNUM
            CLR     C
            SUBB    A,#04H              ;设置值域（十位小于 4）
            JC      PAS_D
            MOV     A,KEYCNT
            DEC     A
            MOV     KEYCNT,A
            CLR     KEYPRES
```



```
            JMP     DAY_L
PAS_D:      MOV     A,DAY           ;输入数据有效，设置日的十位
            ANL     A,#0FH
            MOV     DAY,A
            MOV     A,KEYNUM
            SWAP    A
            ORL     A,DAY
            MOV     DAY,A
            LCALL   SET1302
            CLR     KEYPRES
DAY_L:      MOV     A,KEYCNT        ;设置日的个位
            CJNE    A,#06H,WEE_H
            MOV     A,DAY
            ANL     A,#0F0H
            CJNE    A,#30H,PAS_DL
            MOV     A,KEYNUM
            CLR     C
            SUBB    A,#03H          ;十位为 3 时个位小于 3
            JC      PAS_DL
            MOV     A,KEYCNT
            DEC     A
            MOV     KEYCNT,A
            CLR     KEYPRES
            JMP     WEE_H
PAS_DL:     MOV     A,DAY           ;输入数据有效，设置日的个位
            ANL     A,#0F0H
            MOV     DAY,A
            MOV     A,KEYNUM
            ORL     A,DAY
            MOV     DAY,A
            LCALL   SET1302
            CLR     KEYPRES
            SETB    WEE_S
            CLR     DAY_S
            CLR     MON_S
            CLR     YEA_S
WEE_H:      MOV     A,KEYCNT        ;设置星期，只有个位有效
            CJNE    A,#07H,WEE_L
            MOV     A,WEEK
            ANL     A,#0FH
            MOV     WEEK,A
            MOV     A,KEYNUM
            SWAP    A
            ORL     A,WEEK
            MOV     WEEK,A
            LCALL   SET1302
            CLR     KEYPRES
WEE_L:      MOV     A,KEYCNT
            CJNE    A,#08H,XIANSHI2
            MOV     A,WEEK
            ANL     A,#0F0H
            MOV     WEEK,A
            MOV     A,KEYNUM
            ORL     A,WEEK
            MOV     WEEK,A
            LCALL   SET1302
            CLR     KEYPRES
            CLR     SETFLAG
```

```
            MOV     KEYCNT,#00H
            CLR     WEE_S
            CLR     DAY_S
            CLR     MON_S
            CLR     YEA_S
;日期的显示模式
XIANSHI2:   MOV     A,WEEK              ;显示星期
            ANL     A,#0FH
            MOV     DPTR,#NUMLAB
            MOVC    A,@A+DPTR
            MOV     SHOW1,A
            MOV     A,WEEK
            SWAP    A
            ANL     A,#0FH
            MOV     DPTR,#NUMLAB
            MOVC    A,@A+DPTR
            MOV     SHOW2,A
            MOV     C,WEE_S             ;设置该位时的闪烁处理
            JNC     DAYMOD
            MOV     C,FLASH
            JNC     DAYMOD
            MOV     SHOW1,#00H
            MOV     SHOW2,#00H
DAYMOD:     MOV     A,DAY               ;显示日
            ANL     A,#0FH
            MOV     DPTR,#NUMLAB
            MOVC    A,@A+DPTR
            ORL     A,#80H
            MOV     SHOW3,A
            MOV     A,DAY
            SWAP    A
            ANL     A,#0FH
            MOV     DPTR,#NUMLAB
            MOVC    A,@A+DPTR
            MOV     SHOW4,A
            MOV     C,DAY_S             ;设置该位时的闪烁处理
            JNC     MONMOD
            MOV     C,FLASH
            JNC     MONMOD
            MOV     SHOW3,#00H
            MOV     SHOW4,#00H
MONMOD:     MOV     A,MONTH             ;显示月
            ANL     A,#0FH
            MOV     DPTR,#NUMLAB
            MOVC    A,@A+DPTR
            ORL     A,#80H
            MOV     SHOW5,A
            MOV     A,MONTH
            SWAP    A
            ANL     A,#0FH
            MOV     DPTR,#NUMLAB
            MOVC    A,@A+DPTR
            MOV     SHOW6,A
            MOV     C,MON_S             ;设置该位时的闪烁处理
            JNC     YEAMOD
            MOV     C,FLASH
            JNC     YEAMOD
            MOV     SHOW5,#00H
```

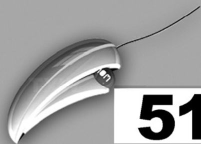


```
                MOV     SHOW6,#00H
YEAMOD:         MOV     A,YEAR              ;显示年
                ANL     A,#0FH
                MOV     DPTR,#NUMLAB
                MOVC    A,@A+DPTR
                ORL     A,#80H
                MOV     SHOW7,A
                MOV     A,YEAR
                SWAP    A
                ANL     A,#0FH
                MOV     DPTR,#NUMLAB
                MOVC    A,@A+DPTR
                MOV     SHOW8,A
                MOV     C,YEA_S             ;设置该位时的闪烁处理
                JNC     DISP
                MOV     C,FLASH
                JNC     DISP
                MOV     SHOW7,#00H
                MOV     SHOW8,#00H
DISP:           ACALL   PLAY                ;调用显示子程序
                AJMP    CLOCK
/*********************************************/
/*          写 DS1302 时间、日期值           */
/* 说明：   设置 DS1302 初始时间，并启动计时 */
/*********************************************/
SET1302:        CLR     T_RST
                CLR     T_CLK
                SETB    T_RST
                MOV     B, #8EH             ;控制寄存器
                LCALL   WRBYTE
                MOV     B, #00H             ;写操作前 WP=0
                LCALL   WRBYTE
                SETB    T_CLK
                CLR     T_RST
                MOV     R0, #Second
                MOV     R7, #7
                MOV     R1, #80H            ;秒写地址
S13021:         CLR     T_RST
                CLR     T_CLK
                SETB    T_RST
                MOV     B, R1               ;写秒、分、时、日、月、星期、年地址
                LCALL   WRBYTE
                MOV     A, @R0              ;写秒数据
                MOV     B, A
                LCALL   WRBYTE
                INC     R0
                INC     R1
                INC     R1
                SETB    T_CLK
                CLR     T_RST
                DJNZ    R7, S13021
                CLR     T_RST
                CLR     T_CLK
                SETB    T_RST
                MOV     B, #8EH             ;控制寄存器
                LCALL   WRBYTE
                MOV     B, #80H             ;控制，WP=1，写保护
```

```
                LCALL    WRBYTE
                SETB     T_CLK
                CLR      T_RST
                RET
/*******************************************/
/*          读 DS1302 时间、日期值          */
/* 说明：   从 DS1302  读时间               */
/*******************************************/
GET1302:        MOV      R0, #Second
                MOV      R7, #7
                MOV      R1, #81H              ;秒地址
G13021:         CLR      T_RST
                CLR      T_CLK
                SETB     T_RST
                MOV      B, R1                 ;秒、分、时、日、月、星期、年地址
                LCALL    WRBYTE
                LCALL    RDBYTE
                MOV      @R0, A                ;秒
                INC      R0
                INC      R1
                INC      R1
                SETB     T_CLK
                CLR      T_RST
                DJNZ     R7, G13021
                RET
/*******************************************/
/*          写 1302 一字节 （内部子程序）   */
/* 说明：        无                          */
/*******************************************/
WRBYTE:         MOV      R4, #8
Inbit1:         MOV      A, B
                RRC      A
                MOV      B, A
                MOV      T_IO, C
                SETB     T_CLK
                CLR      T_CLK
                DJNZ     R4, Inbit1
                RET
/*******************************************/
/*          读 1302 一字节 （内部子程序）   */
/* 说明：        无                          */
/*******************************************/
RDBYTE:         MOV      R4, #8
Outbit1:        MOV      C, T_IO
                RRC      A
                SETB     T_CLK
                CLR      T_CLK
                DJNZ     R4, Outbit1
                RET
/*******************************************/
/*            定时器 1 中断服务              */
/* 说明：                                    */
/*******************************************/
TIME1:          MOV      TH1,#TIMEH_C
                MOV      TL1,#TIMEL_C
                MOV      C,CLKEN
                JNC      BACK
```



```
                MOV     C,RING
                JNC     BACK
                CPL     P1.3
BACK:           DJNZ    10H,RETURN0
                MOV     10H,#240
                CPL     FLASH
                MOV     C,WORN
                JNC     RETURN0
                CPL     RING
RETURN0:        RETI
/*****************************************/
/*          键盘扫描程序                    */
/* 说明：   使用键盘时，要将跳线跳至右端      */
/*****************************************/
SCAN:           MOV     R0,#04H
                MOV     R1,#80H
SCLOP:          MOV     A,R1                    ;扫描部分
                CPL     A
                MOV     DPTR,#PORTC
                MOVX    @DPTR,A
                ANL     A,#0F0H
                MOV     R2,A
                MOVX    A,@DPTR
                ANL     A,#0FH
                CJNE    A,#0FH,STORE            ;判断是否有按键按下
                JMP     NEXT
STORE:          MOV     KEYNUM,A                ;存储扫描结果
                MOV     A,R2
                ORL     A,KEYNUM
                MOV     KEYNUM,A
                ACALL   D1MS
                MOVX    A,@DPTR
                ANL     A,#0FH
                CJNE    A,#0FH,STORE            ;等待按键释放
                MOV     A,KEYCNT
                INC     A
                MOV     KEYCNT,A
                SETB    KEYPRES
                JMP     HANDLE
NEXT:           MOV     A,R1
                RR      A
                MOV     R1,A
                DJNZ    R0,SCLOP
                MOV     KEYNUM,#00H
                JMP     EXIT
HANDLE:         MOV     A,KEYNUM
                        ;按键处理部分，返回按键对应的十六进制数
                MOV     KEYNUM,#00H
IF1:            CJNE    A,#0EEH,IF2
                MOV     KEYNUM,#01H
                JMP     EXIT
IF2:            CJNE    A,#0EDH,IF3
                MOV     KEYNUM,#02H
                JMP     EXIT
IF3:            CJNE    A,#0EBH,IF4
                MOV     KEYNUM,#03H
                JMP     EXIT
IF4:            CJNE    A,#0E7H,IF5
```




```
                MOV     KEYNUM,#04H
                JMP     EXIT
IF5:            CJNE    A,#0DEH,IF6
                MOV     KEYNUM,#05H
                JMP     EXIT
IF6:            CJNE    A,#0DDH,IF7
                MOV     KEYNUM,#06H
                JMP     EXIT
IF7:            CJNE    A,#0DBH,IF8
                MOV     KEYNUM,#07H
                JMP     EXIT
IF8:            CJNE    A,#0D7H,IF9
                MOV     KEYNUM,#08H
                JMP     EXIT
IF9:            CJNE    A,#0BEH,IF0
                MOV     KEYNUM,#09H
                JMP     EXIT
IF0:            CJNE    A,#0BDH,IFA
                MOV     KEYNUM,#00H
                JMP     EXIT
IFA:            CJNE    A,#0BBH,IFB
                MOV     KEYNUM,#0AH
                JMP     EXIT
IFB:            CJNE    A,#0B7H,IFC
                MOV     KEYNUM,#0BH
                JMP     EXIT
IFC:            CJNE    A,#7EH,IFD
                MOV     KEYNUM,#0CH
                JMP     EXIT
IFD:            CJNE    A,#7DH,IFE
                MOV     KEYNUM,#0DH
                JMP     EXIT
IFE:            CJNE    A,#7BH,IFF
                MOV     KEYNUM,#0EH
                JMP     EXIT
IFF:            CJNE    A,#77H,EXIT
                MOV     KEYNUM,#0FH
EXIT:           RET
/*********************************************/
/*              显示子程序                    */
/* 说明：           无                          */
/*********************************************/
PLAY:           MOV     R0,#08H
                MOV     R1,#SHOW1               ;取显示码
                MOV     R2,#80H
DPLOP:          MOV     A,@R1
                MOV     DPTR,#PORTA
                MOVX    @DPTR,A                 ;送出个位的 7 段代码
                MOV     DPTR,#PORTB
                MOVX    @DPTR,A                 ;开相应的位显示
                ACALL   D1MS                    ;显示 162μs
                MOV     DPTR,#PORTB
                MOV     A,#00H
                MOVX    @DPTR ,A                ;关闭十位显示，防止鬼影
                MOV     A,R2
                MOV     A,R2
                RR      A
                MOV     R2,A
```

```
                INC     R1
                DJNZ    R0,DPLOP                ;循环执行 8 次
                RET
/*******************************************/
/*              延时程序                   */
/* 说明：     用于数码管显示延时           */
/*******************************************/
D1MS:           MOV     R6,#150
                DJNZ    R6,$
                RET
NUMLAB: DB      3FH,06H,5BH,4FH,66H,6DH,7DH,07H,7FH,6FH
                END
```

（2）C 语言程序

```
/*********************************************************/
/*     程序名称：数字电子钟                               */
/*     程序功能：显示时间及日期，并可以对时间和日期进行设置。*/
/*               另外，具有闹钟功能，可以手动设置闹钟时间。 */
/*********************************************************/
#include<reg51.h>
#define uchar unsigned char                         //数据类型声明
#define uint unsigned int
// ----------- 全局常量定义 --------------------
#define         TRUE            1                   //布尔量声明
#define         FALSE           0

#define     TIMEMODE        0x11                    //定时器 1 提供蜂鸣器振荡时钟
#define       TIME1msH_C        0xFF                //1ms 定时的加载值高字节
#define       TIME1msL_C        0x00                // 1ms 定时的加载值低字节
#define     PORTA           0x8FFF                  //8255A 口地址
#define     PORTB           0x9FFF                  //8255B 口地址
#define     PORTC           0xAFFF                  //8255C 口地址
#define     CADDR           0xBFFF                  //8255 控制字地址

// ----------- 全局变量定义 --------------------
bit   flash;                                    //设置时间时，当前设置位的闪烁标志位
bit   cl_flash;                                 //预设点报时，蜂鸣器的间歇工作标志位
bit   timemode;                                 //日期、时间切换标志位
bit   timeset;                                  //设置时间标志位
bit   secset,minset,hrset;                      //秒、分、时设置标志位
bit   dayset,mnset,ysset,weset;                 //日、月、年、星期设置标志位
bit   clkenable;                                //报时使能标志位
bit   clkset;                                   //设置报时时间标志位
bit   clkflag;                                  //报时标志位
bit   if_keypress;                              //按键侦听标志位

uchar g_b1msFlag,g_b3msFlag;
```

```
uchar sec,min,hr;                               //秒、分、时存储单元
uchar day,mn,ys,we;                             //日、月、年、星期存储单元
uchar clk_sec,clk_min,clk_hr;                   //报时预设时间存储单元

uchar keypresscnt;                              //按键计数器

sbit P1_3=0x93;                                 /*对 DS1302 操作的相关寄存器声明*/
sbit clk=P1^6;                                  //通信端口声明
sbit dat=P1^5;
sbit rst=P1^4;
                                                //运算器 A 各位声明
sbit A0=ACC^0;
sbit A1=ACC^1;
sbit A2=ACC^2;
sbit A3=ACC^3;
sbit A4=ACC^4;
sbit A5=ACC^5;
sbit A6=ACC^6;
sbit A7=ACC^7;
/*对 DS1302 操作的相关函数声明*/
void InputByte(uchar dd);                   //写 DS1302 一字节
uchar OutputByte(void);                     //读 DS1302 一字节
void Write(uchar addr,uchar num);           //写 DS1302
uchar Read(uchar addr);                     //读 DS1302
uchar ReadSec();                            //读秒
uchar ReadMin();                            //读分钟
uchar ReadHr();                             //读小时
uchar ReadWe();                             //读星期
uchar ReadDay();                            //读日
uchar ReadMn();                             //读月
uchar ReadYs();                             //读年
void WriteSec(uchar num);                   //写秒
void WriteMin(uchar num);                   //写分钟
void WriteHr(uchar num);                    //写小时
void WriteDay(uchar num);                   //写日
void WriteMn(uchar num);                    //写月
void WriteWe(uchar num);                    //写星期
void WriteYs(uchar num);                    //写年
void DisableWP(void);                       //允许写 DS1302
void EnableWP(void) ;                       //进制写 DS1302

void Printer(void);                         /*操作系统的相关函数声明*/
void WriteData(uint Addr,uchar Data);       //写外部寄存器一字节
uchar ReadData(unsigned int Addr);          //读外部寄存器一字节
void Delay(void);                           //延时子程序
void Ini(void);                             //初始化程序
```

```
unsigned char Keyboard(void);                    //按键扫描程序
unsigned char Handlekey(void);                   //按键处理程序

code uchar word[10]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f}; //0～9 的字型码
uint show[8];                                    //8 位 8 段数码管的字形存储单元

// ----------- 主程序 --------------------
void main(void)
{
    uchar Keynum;                                //按键值变量

    Ini();
    DisableWP();
    EnableWP();

    clk_sec=0x00;                                //初始化定时闹钟
    clk_min=0x00;
    clk_hr=0x00;

    while(1)
    {
        Keynum=Handlekey();                      //按键处理

        switch(Keynum)                           //功能按键 a、b、c、d、e、f 处理
        {
            case 0x0a:                           //时间、日期切换按键
              timemode=~timemode;                //时间模式控制位取反
              timeset=0;
              break;

            case 0x0b:                           //设置按键
              timeset=~timeset;                  //时间设置位取反
              keypresscnt=0;
              break;

            case 0x0c:                           //时间、闹钟切换按键
              clkset=~clkset;                    //闹钟设置标志位取反
              timemode=1;
              timeset=0;
              keypresscnt=0;
              break;

            case 0x0d:                           //闹钟使能按键
              clkenable=~clkenable;              //闹钟使能标志位取反
              break;
```



```
        default:
            break;
    }

    if(timemode)                        //判断是否是时间显示模式
    {
    //时间模式
        /*小时显示部分*/
        if(clkset)                      //判断是否是闹钟设置模式
                hr=clk_hr;
        else
              hr=ReadHr();    //读出的数据为压缩 BCD 码，
                              //高四位为个位 BCD 码，低四位为十位 BCD 码

        if(hrset&&flash)      //处于设置模式下，当前设置位闪烁处理
        {
                show[0]=0x00;
                show[1]=0x00;
        }
        else
        {
              show[0]=word[((hr>>4)&0x0f)];
              show[1]=word[(hr&0x0f)];
        }

        show[2]=0x40;
        /*分钟显示部分*/
        if(clkset)                              //判断是否是闹钟设置模式
              min=clk_min;
        else
              min=ReadMin();                    //读出的数据为压缩 BCD 码，
                                    //高四位为个位 BCD 码，低四位为十位 BCD 码

        if(minset&&flash)                       //处于设置模式下，当前设置位闪烁处理
        {
                show[3]=0x00;
                show[4]=0x00;
        }
        else
        {
                show[3]=word[((min>>4)&0x0f)];
                show[4]=word[(min&0x0f)];
        }
        /*秒显示部分*/
        if(clkset)                              //判断是否是闹钟设置模式
        {
```



```
        if(clkenable)      //判断是否闹钟使能
            show[5]=0x00;
        else
            show[5]=0x01;

        show[6]=0x58;
        show[7]=0x38;
    }
    else
    {
        show[5]=0x40;
        sec=ReadSec();           //读出的数据为压缩BCD码，
                                 //高四位为个位BCD码，低四位为十位BCD码
        if(secset&&flash)        //处于设置模式下，当前设置位闪烁处理
        {
            show[6]=0x00;
            show[7]=0x00;
        }
        else
        {
          show[6]=word[((sec>>4)&0x0f)];
          show[7]=word[(sec&0x0f)];
        }
    }

    if(timeset)                  //判断是否是时间设置
    {
        switch(keypresscnt)      //时间模式下的六次按键处理
                                 //小时、分钟、秒各需设置两位
        {
            case 0:
                hrset=1;
                break;
            case 1:                    //设置小时的十位
                if(if_keypress)        //防止再次快速进入，防止hour被修改
                                       //只有在按下按键后，才对按键进行处理
                {
                    if(Keynum>2)           //设置值范围限制
                        keypresscnt=0;
                    else
                    {
                        if(clkset)     //判断是否处于时间设置模式
                            clk_hr=(clk_hr&0x0f)|(Keynum<<4);
                        else
                        {
                            hr=(hr&0x0f)|(Keynum<<4);
```

```
                        WriteHr(hr);
                    }
                }
            }
            if_keypress=0;
            break;
        case 2:                     //设置小时的个位
            if(if_keypress)         //防止再次快速进入，防止 hour 被修改
                                    //只有在按下按键后，才对按键进行处理
            {
                if((Keynum>3)&&((ReadHr()&0xf0) ==0x20))
                    keypresscnt=1;
                else
                {
                    if(clkset)   //判断是否处于时间设置模式
                        clk_hr=(clk_hr&0xf0)|Keynum;
                    else
                    {
                        hr=(hr&0xf0)|Keynum;
                        WriteHr(hr);
                    }
                    hrset=0;
                    minset=1;
                }
            }
            if_keypress=0;
            break;
        case 3:                     //设置分钟的十位
            if(if_keypress)
                                    //防止再次快速进入，防止 min 被修改
                                    //只有在按下按键后，才对按键进行处理
            {
                if(Keynum>5)   //设置值限制
                    keypresscnt=2;
                else
                {
                    if(clkset) //判断是否处于闹钟设置模式
                        clk_min=(clk_min&0x0f)|(Keynum<<4);
                    else
                    {
                        min=(min&0x0f)|(Keynum<<4);
                        WriteMin(min);
                    }
                }
            }
            if_keypress=0;
```



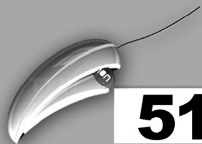
```
            break;
    case 4:                    //设置分钟的个位
        if(if_keypress)   //防止再次快速进入，防止 min 被修改
        {
            if(clkset)        //判断是否处于闹钟设置模式
            {
                clk_min=(clk_min&0xf0)|Keynum;
                minset=0;
                timeset=~timeset;
            }
            else
            {
                min=(min&0xf0)|Keynum;
                WriteMin(min);
                minset=0;
                secset=1;
            }
        }
        if_keypress=0;
        break;
    case 5:                          //设置秒的十位
        if(if_keypress)        //防止再次快速进入，防止 sec 被修改
        {
            if(Keynum>5)
                keypresscnt=4;
            else
            {
                sec=(sec&0x0f)|(Keynum<<4);
                WriteSec(sec);
            }
        }
        if_keypress=0;
        break;
    case 6:                           //设置秒的个位
        if(if_keypress)          //防止再次快速进入，防止 sec 被修改
        {
            sec=(sec&0xf0)|Keynum;
            WriteSec(sec);
            secset=0;
        }
        timeset=~timeset;   //设置完毕，设置时间标志位取反，退出设置
        if_keypress=0;
        break;
    default:
        break;
}
```

```
        }
        else
        {
                hrset=0;
                minset=0;
                secset=0;
        }
}
//日期模式
else
{
      /*年部分*/
        ys=ReadYs();        //读出的数据为压缩 BCD 码，
                            //高四位为个位 BCD 码，低四位为十位 BCD 码
        if(ysset&&flash)    //处于设置模式下，当前设置位闪烁处理
        {
                show[0]=0x00;
                show[1]=0x80;
        }
        else
        {
              show[0]=word[((ys>>4)&0x0f)];
              show[1]=word[(ys&0x0f)]|0x80;
        }
        /*月部分*/
        mn=ReadMn();                //读出的数据为压缩 BCD 码，
                                    //高四位为个位 BCD 码，低四位为十位 BCD 码
        if(mnset&&flash)            //处于设置模式下，当前设置位闪烁处理
        {
                show[2]=0x00;
                show[3]=0x80;
        }
        else
        {
             show[2]=word[((mn>>4)&0x0f)];
             show[3]=word[(mn&0x0f)]|0x80;
        }
        /*日部分*/
        day=ReadDay();              //读出的数据为压缩 BCD 码，
                                    //高四位为个位 BCD 码，低四位为十位 BCD 码
        if(dayset&&flash)           //处于设置模式下，当前设置位闪烁处理
        {
                show[4]=0x00;
                show[5]=0x80;
        }
        else
```



```
{
    show[4]=word[((day>>4)&0x0f)];
    show[5]=word[(day&0x0f)]|0x80;
}
/*星期部分*/
we=ReadWe();             //读出的数据为压缩 BCD 码，
                         //高四位为个位 BCD 码，低四位为十位 BCD 码
if(weset&&flash)         //处于设置模式下，当前设置位闪烁处理
{
      show[6]=0x00;
      show[7]=0x00;
}
else
{
      show[6]=word[((we>>4)&0x0f)];
      show[7]=word[(we&0x0f)];
}

if(timeset)          //判断是否处于设置模式
{
      switch(keypresscnt)
          //日期设置下的八次按键处理（年、月、日、星期各设置两位）
      {
          case 0:
              ysset=1;
              break;
          case 1:                    //设置年的十位
              if(if_keypress)        //防止再次快速进入，防止 ys 被修改
              {
                  ys=(ys&0x0f)|(Keynum<<4);
                  WriteYs(ys);
              }
              if_keypress=0;
              break;
          case 2:                    //设置年的个位
              if(if_keypress)        //防止再次快速进入，防止 ys 被修改
              {
                  ys=(ys&0xf0)|Keynum;
                  WriteYs(ys);
                  ysset=0;
                  mnset=1;
              }
              if_keypress=0;
              break;
          case 3:                    //设置月的十位
              if(if_keypress)        //防止再次快速进入，防止 mn 被修改
```




```
            {
                if(Keynum>1)        //设置值范围限制
                    keypresscnt=2;
                else
                {
                   mn=(mn&0x0f)|(Keynum<<4);
                   WriteMn(mn);
                }
            }
            if_keypress=0;
            break;
        case 4:                         //设置月的个位
            if(if_keypress)         //防止再次快速进入，防止 mn 被修改
            {
                if((Keynum>2)&&((ReadMn()&0xf0) = =0x10))
                  //设置值范围限制（如果十位等于 1，且个位大于 2 ）
                    keypresscnt=3;
                else
                {
                    mn=(mn&0xf0)|Keynum;
                    WriteMn(mn);
                    mnset=0;
                    dayset=1;
                }
            }
            if_keypress=0;
            break;
        case 5:                          //设置日的十位
            if(if_keypress)          //防止再次快速进入，防止 day 被修改
            {
               if(Keynum>3)    //设置值范围限制
                    keypresscnt=4;
                else
                {
                    day=(day&0x0f)|(Keynum<<4);
                    WriteDay(day);
                }
            }
            if_keypress=0;
            break;
        case 6:                     //设置日的个位
            if(if_keypress)     //防止再次快速进入，防止 day 被修改
            {
               if((Keynum>2)&&((ReadDay()&0xf0) = =0x30))
                   //设置值范围限制（如果十位等于 3，且个位大于 2）
```



```
                        keypresscnt=5;
                    else
                    {
                        day=(day&0xf0)|Keynum;
                        WriteDay(day);
                        dayset=0;
                        weset=1;
                    }
                }
                if_keypress=0;
                break;
            case 7:                          //设置星期
                if(if_keypress)              //防止再次快速进入，防止 we 被修改
                {                            //该位只能写 0
                    if(Keynum>0)
                        keypresscnt=6;
                    else
                    {
                        we=(we&0x0f)|(Keynum<<4);
                        WriteWe(we);
                    }
                }
                if_keypress=0;
                break;
            case 8:                     //设置星期的个位
                if(if_keypress)         //防止再次快速进入，防止 we 被修改
                {
                    if((Keynum<1)||(Keynum>7))
                                        //设置值范围限制（如果大于 7 或小于 1）
                        keypresscnt=7;
                    else
                    {
                        we=(we&0xf0)|Keynum;
                        WriteWe(we);
                        weset=0;
                        timeset=~timeset;
                    }
                }
                if_keypress=0;
                break;
            default:
                break;
        }

    }
    else
```

```
                {
                    ysset=0;
                    mnset=0;
                    dayset=0;
                    weset=0;
                }
            }
            Printer();              //调用显示程序

            if(clkenable&&(ReadHr()==clk_hr)&&(ReadMin()==clk_min))
                    //是否到报时时间（闹钟使能开，且当前分钟与小时值与设置值相等）
                clkflag=1;
            else
                clkflag=0;
    }
}
/********************************************/
/*              初始化程序                    */
/* 说明：  配置寄存器及初始化参数               */
/********************************************/
void Ini(void)
{
        timemode=0;                         //各个标志位初始化
        timeset=0;
        secset,minset,hrset=0;
        dayset,mnset,ysset,weset=0;
        clkset=0;
        clkenable=1;
        g_b1msFlag=0;
        flash=1;
        clkflag=0;

        if_keypress=0;
        keypresscnt=0;

        TMOD= TIMEMODE;                     // 定时、计数器初始化
        TH0 = TIME1msH_C;
        TL0 = TIME1msH_C;
        TR0 = TRUE;

        ET0 = 1;                            // 此处只允许 T0 中断
        EA=1;
                                            // 8255 工作模式
        WriteData(CADDR,0x81);
        WriteData(PORTC,0xff);
}
```



```
/*********************************************/
/*                总线协议程序                    */
/* 说明：            无                          */
/*********************************************/
//写一字节到 DS1302
void InputByte(uchar dd)
{
        uchar i;
        ACC=dd;
        EA=0;                        // 保证该过程不被中断
        for(i=8;i>0;i--)             // 共 8 次
        {
          dat=A0;
          clk=1;
          clk=0;
          ACC=ACC>>=1;
        }
         EA=1;
}
//从 DS1302 中读一字节
uchar OutputByte(void)
{
        uchar i;
        dat=1;
        EA=0;                        // 保证该过程不被中断
        for(i=8;i>0;i--)
        {
            ACC=ACC>>1;
            A7=dat;
            clk=1;
            clk=0;
        }
         EA=1;
        return(ACC);
}
//写 DS1302 寄存器
void Write(uchar addr,uchar num)
{
      rst=0;
      clk=0;
      rst=1;
      InputByte(addr);          // address
      InputByte(num);           // data
      clk=1;
      rst=0;
```

```
    }
//读 DS1302 寄存器
uchar Read(uchar addr)
{
     uchar dd=0;
     rst=0;
     clk=0;
     rst=1;
     InputByte(addr);
     dd=OutputByte();
     clk=1;
     rst=0;
     return(dd);
    }
//读秒寄存器
uchar ReadSec()
{
    uchar dd;
    dd=Read(0x81);
    return(dd);
    }
//读分钟寄存器
uchar ReadMin()
{
    uchar dd;
    dd=Read(0x83);
    return(dd);
     }
//读小时寄存器
uchar ReadHr()
{
     uchar dd;
     dd=Read(0x85);
     return(dd);
    }
//读星期寄存器
uchar ReadWe()
{
     uchar dd;
     dd=Read(0x8b);
     return(dd);
     }
//读日寄存器
uchar ReadDay()
{
     uchar dd;
     dd=Read(0x87);
```

```
        return(dd);
      }
//读月寄存器
uchar ReadMn()
{
        uchar dd;
        dd=Read(0x89);
        return(dd);
      }
//读年寄存器
uchar ReadYs()
{
        uchar dd;
        dd=Read(0x8d);
        return(dd);
      }
//写秒寄存器
void WriteSec(uchar num)
{
        Write(0x80,num);
}
//写分钟寄存器
void WriteMin(uchar num)
{
        Write(0x82,num);
}
//写小时寄存器
void WriteHr(uchar num)
{
        Write(0x84,num);
}
//写日寄存器
void WriteDay(uchar num)
{
      Write(0x86,num);
}
//写月寄存器
void WriteMn(uchar num)
{
        Write(0x88,num);
      }
//写星期寄存器
void WriteWe(uchar num)
{
      Write(0x8a,num);
      }
```



```
//写年寄存器
void WriteYs(uchar num)
{
    Write(0x8c,num);
  }
//写控制字，允许写操作
void DisableWP(void)
{
   Write(0x8e,0x00);
}
//写保护，不允许写
void EnableWP(void)
{
   Write(0x85,0x80);
}
/********************************************/
/*                 显示子程序                  */
/* 说明：            无                        */
/********************************************/
void Printer(void)
{
     int i;
     uchar MASK,mask=0x01;
     for(i=0;i<8;i++)                    //扫描显示 8 位数码管
     {
        MASK=mask;                       //位屏蔽码
          mask=mask<<1;

        WriteData(PORTA,show[i]);        //输出字形
        WriteData(PORTB,MASK);
        Delay();
        WriteData(PORTA,0x00);           //输出屏蔽位
        WriteData(PORTB,MASK);
     }
}
/********************************************/
/*                 写外部寄存器                */
/* 说明：            无                        */
/********************************************/
void WriteData(uint Addr,uchar Data)
{
    *((uchar xdata *)Addr)=Data;
}
/********************************************/
/*                 读外部寄存器                */
/* 说明：            无                        */
```



```
/*********************************************/
unsigned char ReadData(unsigned int Addr)
{
    return *((unsigned char xdata *)Addr);
}
/*********************************************/
/*                延时程序                   */
/* 说明：          无                        */
/*********************************************/
void Delay(void)
{
    int i,j;
    for(i=0;i<20;i++)
        for(j=0;j<10;j++);

}

/*********************************************/
/*             键盘扫描程序                  */
/* 说明：      使用键盘时，要将跳线跳至右端  */
/*********************************************/
unsigned char Keyboard(void)
{
  unsigned char scan,keypress=0x00;
                      //必须初始化，否则当有按键按下时，程序将默认为初值
    unsigned int   i;
    unsigned char MASK,mask=0x10;
    for(i=0;i<4;i++)                          //扫描 4 次键盘
    {
     MASK=～(mask);
          mask=mask<<1;
       WriteData(PORTC,MASK);                 //高 4 位送出屏蔽码
       scan=ReadData(PORTC);                  //低 4 位读回扫描结果
       if((scan&0x0f)!=0x0f)                  //如果有按键按下，则保存结果
       {
           if_keypress=1;                     //按键监视位置 1
           keypresscnt++;                     //按键计数器加 1
           keypress=scan;
           Delay();
       }
       while((ReadData(PORTC)&0x0f)!=0x0f);   //等待按键释放
    }
    return keypress;                          //返回按键扫描码
}
/*********************************************/
/*             键盘读取程序                  */
/* 说明：     返回按下按键的值               */
```

```
/*********************************************/
uchar Handlekey(void)
{
        unsigned char handlekey;
        switch(Keyboard())                                  //读取按键扫描码
        {
            case 0xee:
                    handlekey=0x01;
                    break;
             case 0xed:
                    handlekey=0x02;
                    break;
             case 0xeb:
                    handlekey=0x03;
                    break;
             case 0xe7:
                    handlekey=0x04;
                    break;
             case 0xde:
                    handlekey=0x05;
                    break;
             case 0xdd:
                    handlekey=0x06;
                    break;
             case 0xdb:
                    handlekey=0x07;
                    break;
             case 0xd7:
                    handlekey=0x08;
                    break;
             case 0xbe:
                    handlekey=0x09;
                    break;
             case 0xbd:
                    handlekey=0x00;
                    break;
             case 0xbb:
                    handlekey=0x0a;
                    break;
             case 0xb7:
                    handlekey=0x0b;
                    break;
            case 0x7e:
                    handlekey=0x0c;
                    break;
             case 0x7d:
                    handlekey=0x0d;
```

```
                break;
            case 0x7b:
                handlekey=0x0e;
                break;
            case 0x77:
                handlekey=0x0f;
                break;
            default:
                handlekey=0xff;
                break;
        }
        return handlekey;                                    //返回按键对应的十六进制数据
}
/*******************************************/
/*              定时器 0 中断服务                   */
/* 说明：   1ms 中断一次                           */
/*******************************************/
void   Timer0_Int(void) interrupt 1
{

        TH0 = TIME1msH_C;                                    //重设初值
        TL0 = TIME1msL_C;

        if(clkflag&&cl_flash&&flash)                         //判断是否需要蜂鸣器发声
            P1_3=～P1_3;                                     //P1.3 输出矩形波使蜂鸣器发声

        if(g_b3msFlag==6)
        {
            g_b3msFlag=0;
            flash=～flash;
        }
        if(g_b1msFlag==200)
        {
            g_b1msFlag=0;
            g_b3msFlag++;
            cl_flash=～cl_flash;
        }
        g_b1msFlag++;
}
```



20.4 案例笔记

20.4.1 如何阅读英文的芯片数据手册

芯片数据手册英译为“Datasheet”，它是电子零件开发者或者制造者针对其所生产的电

子零件而出具的详细描述文件。在 Datasheet 中描述了电子零件的各种关键数据。越是复杂的电子零件，其 Datasheet 就越复杂。而某些简单的电子零件则没有 Datasheet（因为它的参数已为绝大多数的使用者所了解）。

电子世界是一个非常复杂的世界，对于电子产品来说，不能只了解电子零件的功能，而不管这个电子零件的适用范围，盲目地去使用这些电子零件。例如，如果把一枚 LED 直接接到 220V 的电源上，将会听到啪的一声轻响，然后冒出一阵青烟，而根本就不会看到 LED 被点亮的效果。正是因为这种复杂性，所以厂商也才会出具标准的 Datasheet 来描述他们的电子产品，否则将无法使用。

本书已讲解了许多芯片：555、8255、ADC0809、DS18B20、DS1302 等，建议读者去查阅一下这些芯片的手册。查阅这些手册并不难，它们都是使用通俗易懂的语句，向使用者交代清楚该产品的特点、功能以及使用方法，运用在大学里所学到的英文知识就已足够。

Datasheet 通常包含如下几部分内容：

1）基本介绍。这部分内容一般在首页中会提及，通常含有大量的一句话信息，从这个部分可以基本判定 Datasheet 所表述的物料是否符合你的要求。

2）实现的功能介绍。这部分内容会标注零件可以实现的功能，以及正常实现功能的物理环境要求，或者适用区间。有些零件还会标注零件使用条件和后续处理，以及零件的最大处理能力。

3）引脚配置。这部分非常详细地介绍这个零件每一个引脚的定义，首先往往是一个图片，上面会标注诸如 VCC、VACC、GND、PA 等标注。然后紧接着是一个列表，标明每一个引脚的具体定义，有时也会把电气参数包括进去。

4）零件工作的电气参数。通过这部分可以了解所描述零件的正常工作电压区间和电流区间，通常情况下，一个电子零件是具有一定范围的“兼容”空间的。而这个零件的“典型”值，就是我们需要关注的内容。如 LED 灯，它通常的典型电流为 20mA。

5）零件的设置参数和设置说明。对于某些电子零件来说，它在使用过程中需要进行各种设置，如三轴加速度传感器，就需要事先配置内部寄存器，使它输出我们需要的数据。从这部分可以找到一个完整的配置清单及说明。

6）零件的封装尺寸。同一个功能的零件，即便电气特性完全相同。由于其应用的环境不同，厂商也会根据实际的市场需求提供不同的封装产品。就如 Arduino UNO 的核心 ATMEGA328 一样，它就有三种标准的封装（TQFP、PDIP、MLF）。

7）其他。在 Datasheet 里面还会包含一些其他的信息，如勘误表、版本历史、典型应用电路、零件测试曲线等信息，这些信息也需要了解。

阅读一个芯片的数据手册需要注意如下问题：

1）先看芯片的特性（Features）、应用场合（Applications）以及内部框图。这有助于对芯片有一个宏观的了解，此时需要弄清楚该芯片的一些比较特殊的功能，充分利用芯片的特殊功能，对整体电路的设计，将会有极大的好处。

2）重点关注芯片的参数，同时可以参考手册给出的一些参数图，这是决定是否采用该芯片的重要依据。

3）选定器件后，研究芯片引脚定义、推荐的 PCB layout，这些都是在硬件设计过程中必须掌握的。所有引脚中，要特别留意控制信号引脚或者特殊信号引脚，这是将来用好该芯片的前提。

4）认真研读芯片内部寄存器，对寄存器的理解程度，直接决定了对该芯片的掌握程度。对于这些寄存器，必须清楚它们上电后的初始值、所能实现的功能、每个 bit 所代表的含义。

5）仔细研究手册给出的时序图，这是对芯片进行正确操作的关键。单个信号的周期、上升时间、下降时间、建立时间、保持时间，以及信号之间的相位关系，所有这些都必须研究透彻。

6）凡是芯片数据手册中的“note”，都必须仔细阅读，一般这都是能否正确使用，或能否用好芯片的关键所在。

▷▷▷ 20.4.2　DS1302 总结

（1）DS1302 的基本特点

1）时钟计数功能，可以对秒、分钟、小时、月、星期、年计数。年计数可达到 2100 年。

2）有 31 字节的额外数据暂存寄存器。

3）最少 I/O 引脚传输，通过三引脚控制。

4）工作电压为 2.0～5.5V。

5）工作电流小于 320nA（2.0V）。

6）读写时钟寄存器或内部 RAM（31 字节的额外数据暂存寄存）可以采用单字节模式和突发模式。

7）8-pin DIP 封装（见图 20-3）或 8-pin SOIC 封装（见图 20-4）。

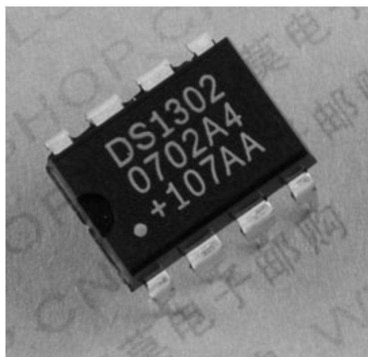


图 20-3　DIP 封装

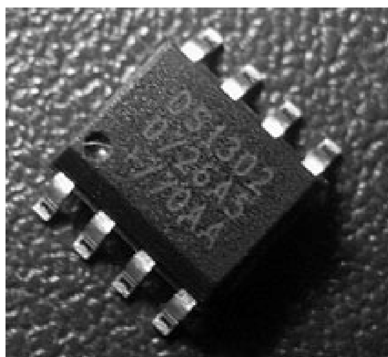


图 20-4　SOIC 封装

8）兼容 TTL（5.0V）。

9）可选的工业级别，工作温度为-40～85℃。

（2）DS1302 的内部结构

DS1302 的内部结构如图 20-5 所示。

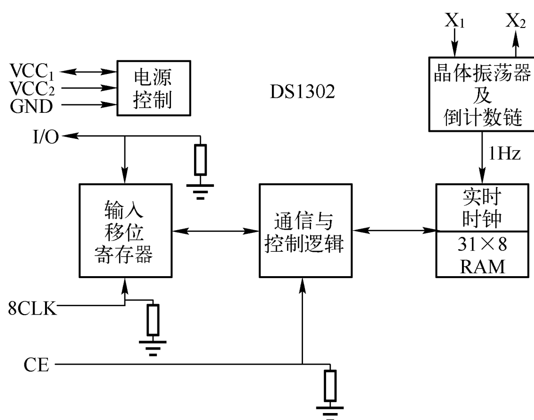


图 20-5　DS1302 的内部结构

（3）DS1302 的功能

DS1302 包括时钟/日历寄存器和 31 字节（8 位）的数据暂存寄存器，数据通信仅通过一条串行 I/O 口。实时时钟/日历提供包括秒、分、时、日期、月份、年份和星期几的信息。闰年可自行调整，可选择 12 小时制和 24 小时制，可以设置 AM、PM。

（4）DS1302 的 SPI 时序

串行外设接口 SPI（Serial Peripheral Interface）总线技术是 Motorola 公司推出的一种同步串行接口。SPI 用于 CPU 与各种外设进行全双工、同步串行通信。它只需四条线就可以完成 MCU 与各种外设的通信，这四条线是串行时钟线（CSK）、主机输入/从机输出数据线（MISO）、主机输出/从机输入数据线（MOSI）、低电平有效从机选择线（CS）。如图 20-6 所示。

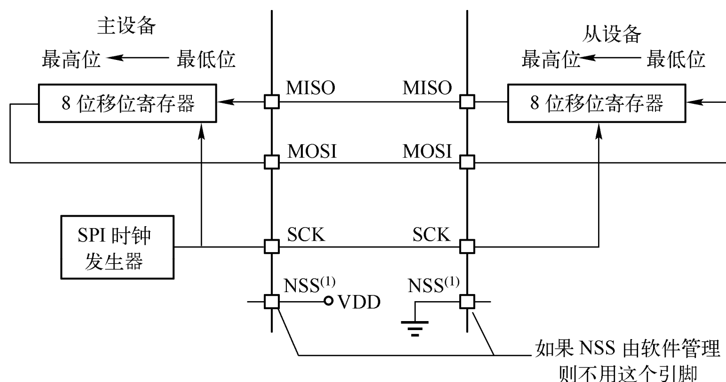


图 20-6　SPI 主从机连接

（5）DS1302 的读写程序

DS1302 与微处理器进行数据交换时，首先由微处理器向电路发送命令字节，命令字节最高位 MSB 必须为逻辑 1，如果为 0，则禁止写 DS1302，即写保护；若 D6 为 0，指定时钟数据，若 D6 为 1，指定 RAM 数据；D5～D1 指定输入或输出的特定寄存器；最低位 LSB 为逻辑 0，指定写操作（输入），若 D0 为 1，指定读操作（输出）。

在 DS1302 的时钟日历或 RAM 进行数据传送时，DS1302 必须首先发送命令字节。若进行单字节传送，8 位命令字节传送结束之后，在下 2 个 SCLK 周期的上升沿输入数据字节，或在下 8 个 SCLK 周期的下降沿输出数据字节。 与 DS1302 通信的接口采用 P1.4 端口、P1.5 端口和 P1.6 端口。

从 DS1302 中读出的时间及日期的数据为压缩 BCD 码，高四位为个位 BCD 码，低四位为十位 BCD 码。所以，只需将读出的数据移位、屏蔽操作后查表即可。

DS1302 是 SPI 总线驱动方式。它不仅要向寄存器写入控制字，还需要读取相应寄存器的数据。

要想与 DS1302 通信，首先要先了解 DS1302 的控制字。DS1302 的控制字如图 20-7 所示。

7	6	5	4	3	2	1	0
1	RAM / $\overline{CK}$	A4	A3	A2	A1	A0	RD / $\overline{WR}$

图 20-7　DS1302 控制字（即地址及命令字节）

控制字的最高有效位（位 7）必须是逻辑 1，如果它为 0，则不能把数据写入 DS1302 中。

位 6：如果为 0，则表示存取日历时钟数据，为 1 表示存取 RAM 数据。

位 5～位 1（A4～A0）：指示操作单元的地址。

位 0（最低有效位）：如为 0，表示要进行写操作，为 1 表示进行读操作。

控制字总是从最低位开始输出。在控制字指令输入后的下一个 SCLK 时钟的上升沿时，数据被写入 DS1302，数据输入从最低位（0 位）开始。同样，在紧跟 8 位的控制字指令后的下一个 SCLK 脉冲的下降沿，读出 DS1302 的数据，读出的数据也是从最低位到最高位。

相应子程序为写一字节到 DS1302 的 InputByte(uchar dd)子程序和从 DS1302 中读一字节的 OutputByte(void)子程序。

案例21



串 行 通 信

▷▷ 21.1 案例任务

打开串口调试助手（选择其他串口接收类软件也可以），如图 21-1 所示，波特率设为 9600，选中十六进制模式，发送 02，它是通信开始标志或称报文头，然后串口调试助手去掉十六进制发送模式，采用默认的 ASCII 码模式，以后发送的数据则为有效数据，在单片机数码第一位显示 r，后面为接收的有效数据，当上位机发送 F，则数码管显示清除，重新从头显示，上位机传送数据可为 0～9 和 A～E。当下位机给上位机发送时，按动键盘，在数码管第一位显示 F，最后一位为发送的字符，同时可在计算机的串口调试助手接收区看到下位机发送的数据。



图 21-1　串口调试助手界面

▷▷ 21.2 案例要点

（1）掌握数据通信中的基本概念

1）通信种类：异步通信、同步通信。

2）串行通信的制式：单工制式、半双工制式、全双工制式。

3）串行通信的校验：奇偶校验、循环冗余码校验（CRC）、累加和校验。

4）波特率（Baud Rate）：传送数据位的速率，一般指每秒钟传送二进制代码的位数。单位为 bit/s。

（2）了解通信协议的概念、51 单片机串口异步通信数据格式

51 单片机串口异步通信数据格式如图 21-2 所示，其各位细节信息见表 21-1。

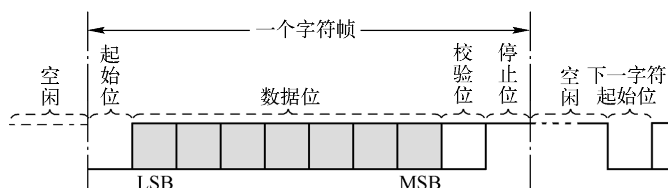


图 21-2　异步通信格式

表 21-1　串行通信数据格式

	起始位	数据位	奇偶校验位	停止位
位数	1	5～8	1	1；1.5；2
是否必须有	必有	必有	可无	必有

（3）掌握 RS-232C 接口标准总线

（4）51 单片机串口的结构及功能

51 单片机串口的结构如图 21-3 所示，可分成三部分：

1）发送电路。

2）接收电路。

3）控制逻辑。

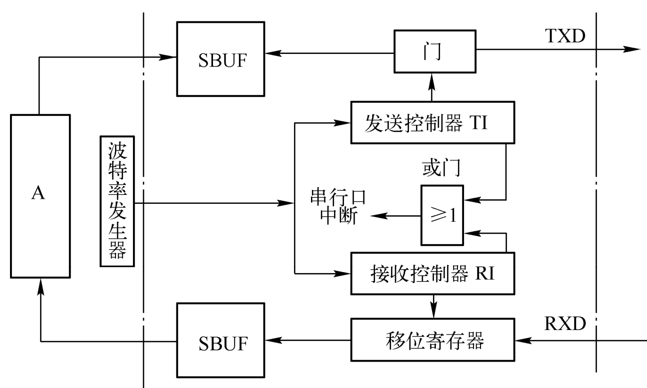


图 21-3　51 单片机串口的结构

（5）51 单片机串口的四种工作方式

1）方式 0：8 位移位寄存器 I/O 方式。

2）方式 1：10 位异步通信方式。

3）方式 2：11 位异步通信方式。

4）方式 3：11 位异步通信方式。

（6）51 单片机串行口工作寄存器 SCON 和 PCON 各位的含义

（7）掌握 51 单片机与 PC 间通信的硬件电路及驱动程序设计

▷▷ 21.3 案例设计

▷▷▷ 21.3.1 硬件电路

串行通信电路如图 21-4 所示。51 单片机可通过两种方式与 PC 相连，一种是通过 MAX3232 芯片与 PC 的串口相连；一种是通过 CH341 芯片与 PC 的 USB 接口相连。

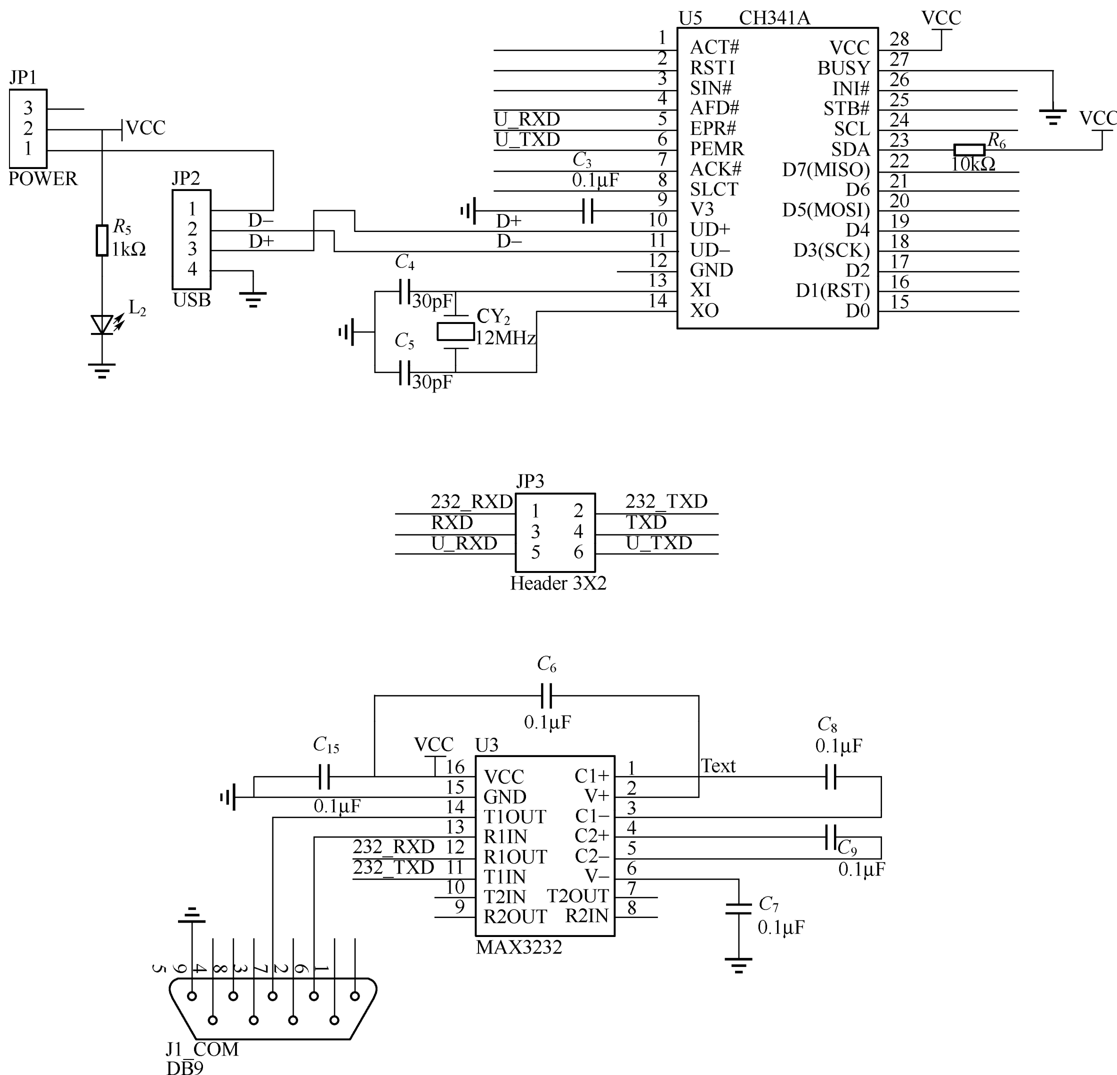


图 21-4 串行通信电路

▷▷▷ 21.3.2 软件代码

```
//变量定义区
```



```
unsigned char xdata porta _at_ 0x8fff;
unsigned char xdata portb _at_ 0x9fff;
unsigned char xdata PORTC _at_ 0xafff;
unsigned char xdata caddr _at_ 0xbfff;
#define       uchar                       unsigned char
#define       unint                       unsigned int
#define       PORTC                           0xafff
#define       CADDR                           0xbfff
bit           if_keypress;
uchar          keypresscnt;
uchar          Keynum;
void           WriteData(unsigned int Addr,unsigned char Data);
uchar          ReadData(unsigned int Addr);
uchar          Keyboard(void);
uchar          Handlekey(void);
unint         show[8];
code uchar    word[16]={0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x41,
                        0x42,0x43,0x44,0x45,0x46};
unsigned char xdata *a _at_ 0x0060;
unsigned char xdata *b _at_   0x0070;
unsigned char xdata *ctr _at_ 0x0075;
static char i,j,jj;
unsigned char tmp,enable;

unsigned char code displaybit[8]={0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01};
unsigned char code displaycode[30]={0x3f,0x06,0x5b,0x4f,
                                    0x66,0x6d,0x7d,0x07,
                                    0x7f,0x6f,0x77,0x7c,
                                    0x39,0x5e,0x79,0x71,0x00,0x77,0x7c,
                                    0x39,0x5e,0x79,0x71,0x76,0x79,0x38,0x38,0x3f,0x31};
unsigned char displaycount;   //记录当前显示到第几位
//unsigned char displaybuf[8]={0,0,18,19,20,21,22,0}; //初始化显示缓存，均为 16，即都不显示，处
                                                //于消隐状态  void display(void)
unsigned char displaybuf[8]={16,16,27,26,25,24,23,16};
//头文件区
#include<AT89X52.h>
//初始化函数区

 void Initfunction8255(void)   //初始化 8255
 {
     ctr=&caddr;
     *ctr=0x81;
     b=&portb;
     a=&porta;
      if_keypress=0;
      keypresscnt=0;
```

```
        Keynum=0;
        j=7;
  }
  void initcom(void)
  {
    TMOD |= 0x20;       //T1 方式 2
    PCON = 0x00;        //波特率不增倍
    SCON=0x50;          //串口工作方式 1
    TH1=0xE6;           //波特率 1200
    TL1=0xE6;
    TR1=1;
    ES=1;               //开中断
  }

  void Initfunctiontimer0(void)       //初始化 T0，定时 1ms
  {
  TMOD|=0x01;
  TR0=1;
  EA=1;
  ET0=1;
  TH0=0xfc;
  TL0=0x18;
  }
//功能函数实现区
 /*********************************************/
/*                 写外部寄存器                   */
/* 说明：            无                          */
/*********************************************/
void WriteData(unint Addr,uchar Data)
{
      *((uchar xdata *)Addr)=Data;
}
/*********************************************/
/*                 读外部寄存器                   */
/* 说明：            无                          */
/*********************************************/
unsigned char ReadData(unsigned int Addr)
{
      return *((unsigned char xdata *)Addr);
}

/*********************************************/
/*               键盘扫描程序                     */
/* 说明： 使用键盘时，要将跳线跳至右端            */
/*********************************************/
unsigned char Keyboard(void)
```



```
{
    unsigned char scan,keypress=0x00; //初始化，否则当有按键按下时，程序将默认为初值
    unsigned int  i;
    unsigned char MASK,mask=0x10;
    for(i=0;i<4;i++)
    {
        MASK=~(mask);
        mask=mask<<1;

        WriteData(PORTC,MASK);
        scan=ReadData(PORTC);
        if((scan&0x0f)!=0x0f)
        {
            if_keypress=1;
            keypresscnt++;
            keypress=scan;
        }
        while((ReadData(PORTC)&0x0f)!=0x0f);

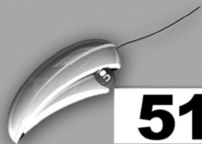
    }
    return keypress;
}

/*******************************************/
/*              键盘读取程序               */
/*  说明：      返回按下按键的值           */
/*******************************************/
uchar Handlekey(void)
{
    unsigned char handlekey;
    switch(Keyboard())
    {
        case 0xee:
            handlekey=0x01;
            break;
        case 0xed:
            handlekey=0x02;
            break;
        case 0xeb:
            handlekey=0x03;
            break;
        case 0xe7:
            handlekey=0x04;
            break;
        case 0xde:
            handlekey=0x05;
```

```
                break;
            case 0xdd:
                handlekey=0x06;
                break;
            case 0xdb:
                handlekey=0x07;
                break;
            case 0xd7:
                handlekey=0x08;
                break;
            case 0xbe:
                handlekey=0x09;
                break;
            case 0xbd:
                handlekey=0x00;
                break;
            case 0xbb:
                handlekey=0x0a;
                break;
            case 0xb7:
                handlekey=0x0b;
                break;
            case 0x7e:
                handlekey=0x0c;
                break;
            case 0x7d:
                handlekey=0x0d;
                break;
            case 0x7b:
                handlekey=0x0e;
                break;
            case 0x77:
                handlekey=0x0f;
                break;
            default:
                handlekey=0xff;
                break;
        }
        return handlekey;
}
 void delay200ms(void)
{
    unsigned char i,j,k;
    for(i=5;i>0;i--)
    for(j=132;j>0;j--)
    for(k=150;k>0;k--);
```



```
}
  void display(void)       //显示函数
  {
      if(displaycount==1)   //个位显示
    {
      *a=displaycode[displaybuf[displaycount]]|0x00;
    }
    else
      {
       *a=displaycode[displaybuf[displaycount]];   //送字型
      }
  *b=displaybit[displaycount];                     //送字位
  displaycount++;
  if(displaycount==8)                              //八位显示完毕后从头显示
    {
      displaycount=0;
    }
  }
void SendD( unsigned char SendDat)
 {
   SBUF=SendDat;
   while(!TI);
    TI=0;
 }
 //主函数区
void main (void)
{
  delay200ms();
  delay200ms();
  Initfunction8255();
  Initfunctiontimer0();
  initcom();
   while(1)
    {
       Keynum=Handlekey();          //按键处理
       if(if_keypress)
         {
              *b=0x00;             //送字位
           for(jj=0;jj<8;jj++)
              {
                displaybuf[jj]=16;
              }
           displaybuf[7]=15;
           displaybuf[0]=(word[Keynum]-0x30);
           SendD(word[Keynum]);
           if_keypress=0;
```




```
                    delay200ms();

                }
            }

        }

        //中断函数区
        void timer0(void ) interrupt 1
          {

              TH0=0xfc;
              TL0=0x18;
              display();
          }
        //串口中断
        void serial(void) interrupt 4
          {

            if (RI)
              {
                RI=0;

                if(enable)
                  {
                   if(SBUF==0x46)
                     {
                       j=7;
                       *b=0x00;          //送字位
                       for(jj=0;jj<7;jj++)
                         {
                           displaybuf[jj]=16;
                         }
                     }
                   else
                     {
                       j--;
                       displaybuf[j]=SBUF-0x30;
                       SendD(displaybuf[j]+0x30);
                        if(j==0)
                         {
                           j=7;
                         }
                     }
                  }
```

```
            if(SBUF==0x02)
            {
                enable=1;
                j=7;
                *b=0x00;                    //送字位
                  for(jj=0;jj<8;jj++)
                    {
                      displaybuf[jj]=16;
                    }
                  displaybuf[7]=28;
            }
        }
    }
```



21.4 案例笔记

21.4.1 SPI、I²C、UART 三种串行总线协议的区别

SPI（串行外设接口）、I^2C（Inter IC Bus）和 UART（Universal Asynchronous Receiver Transmitter，通用异步收发器）主要区别如下：

1）电气信号线上的不同。

① SPI 总线由三条信号线组成：串行时钟（SCLK）、串行数据输出（SDO）、串行数据输入（SDI）。SPI 总线可以实现多个 SPI 设备互相连接。提供 SPI 串行时钟的 SPI 设备为 SPI 主机或主设备（Master），其他设备为 SPI 从机或从设备（Slave）。主从设备间可以实现全双工通信，当有多个从设备时，还可以增加一条从设备选择线。

如果用通用 I/O 口模拟 SPI 总线，必须要有一个输出口（SDO）、一个输入口（SDI），另一个口则根据实现的设备类型而定，如果要实现主从设备，则需输入/输出口，若只实现主设备，则需输出口即可，若只实现从设备，则只需输入口即可。

② I^2C 总线是双向、两线（SCL、SDA）、串行、多主控（Multi-master）接口标准，具有总线仲裁机制，非常适合在器件之间进行近距离、非经常性的数据通信。在它的协议体系中，传输数据时都会带上目的设备的设备地址，因此可以实现设备组网。

如果用通用 I/O 口模拟 I^2C 总线，并实现双向传输，则需一个输入/输出口（SDA），另外还需一个输出口（SCL）。

③ UART 总线是异步串口，因此一般比前两种同步串口的结构要复杂很多，一般由波特率产生器（产生的波特率等于传输波特率的 16 倍）、UART 接收器、UART 发送器组成，硬件上有两根线，一根用于发送，一根用于接收。

显然，如果用通用 I/O 口模拟 UART 总线，则需一个输入口和一个输出口。

2）SPI 和 UART 可以实现全双工，但 I^2C 不行。

3）I^2C 的速度比 SPI 慢一点，协议比 SPI 复杂一点，但是连线比标准的 SPI 要少。

▷▷▷ 21.4.2 串口四种工作方式总结

串口四种工作方式总结见表 21-2。

表 21-2 串口四种方式总结

方式	方式 0	方式 1	方式 2	方式 3
名称	8 位移位寄存器 I/O 方式	10 位异步通信方式	11 位异步通信方式	11 位异步通信方式
一帧数据格式	8 位数据位	起始位0 8位数据位 停止位1	起始位0 8位数据位 校验位 停止位1 发送第 9 位由 SCON 的 TB8 提供； 接收第 9 位由 SCON 的 RB8 提供； 第 9 位可为校验位，可为多机通信的地址/特征位	
波特率	固定 $\frac{f_{osc}}{12}$	可变： $\frac{2^{SMOD}}{64}\times$(T1溢出率) $=\frac{2^{SMOD}}{64}\frac{f_{osc}}{12(256-C)}$	固定： $\frac{2^{SMOD}}{32}f_{osc}$	同方式 1
引脚功能	TXD：输出 $\frac{f_{osc}}{12}$ 同步脉冲 RXD：数据 I/O	TXD：数据输出 RXD：数据输入		
应用	扩展 I/O	串行通信		

案例22



数字计算器



22.1 案例任务

本案例是一个比较综合的项目，本案例的目标是希望读者掌握单片机开发的基本流程和注意一些常见的问题。

本案例实现浮点数（包括正、负）的加、减、乘、除及开根号的功能。

为了解决键盘按键有限的问题，本案例采用按键复用来解决，如图 22-1 所示。第二功能键按一次，发光二极管 6 变亮，表示键 A、B、C、D、E 处于第二功能状态，再按一下按键，发光二极管 6 灭，表示键 A、B、C、D、E 处于第一功能状态。加、减、乘、除也有对应的状态显示发光二极管。当输入一个“和操作”操作符后，操作符对应的状态显示发光二极管变亮，直到输入第二个操作数和等号后操作符状态显示发光二极管才灭。发光二极管 3 对应的操作符是加，发光二极管 4 对应的操作符是减，发光二极管 5 对应的操作符是乘，发光二极管 3 和发光二极管 4 同时亮对应的操作符是除。有了状态显示使用者就可以很容易地判断出当前输入的是第一个还是第二个操作数，并且在输入第二个操作数时能够根据操作符状态显示的发光二极管判断出操作符。开根号只有一个操作数，因此没有必要增加操作符状态显示。考虑到等号是用第二功能键实现的，为了避免使用的不便，设计中当输入两个操作数和一个操作符后，按加、减、乘、除键（即 A、B、C、D 的第一功能键）和按等号键具有同样的效果，都实现等号的功能。

由于仿真板只有 8 位数码管，所以输入的数据或计算结果最多只有 8 位（负数只能有 7 位），如果计算结果超出 8 位（或 7 位）则四舍五入到全屏显示即 8 位（或 7 位）。

22.2 案例要点

（1）8255 外扩接口的硬件电路设计及驱动程序编写

（2）顺序、分支、循环程序的编写

（3）流程图的画法

（4）子程序的编写

1）子程序的优点：

① 对程序的修改可局部进行，其他部分可保持不变。

② 缩短程序长度，程序可读性好，便于功能扩充和版本升级。

③ 单个模块结构的程序功能单一，易于编写、调试和修改。

④ 便于分工，从而可使多个程序员同时进行程序的编写和调试工作，加快软件研制进度。

⑤ 对于使用频繁的子程序可以建立子程序库，便于多个模块调用。

2）子程序具有的七个要素：

① 功能说明。

② 子程序名。

③ 入口地址。

④ 入口参数。入口参数是需要传给子程序的参数。子程序有三种参数传递方式，即寄存器传送参数、存储器传送参数、堆栈传送参数。

⑤ 出口参数。出口参数是子程序送回调用程序的结果参数。

⑥ 占用资源。

⑦ 子程序的调用。

3）编写与使用子程序需要注意以下四点：

① 现场保护与现场恢复，避免与主程序冲突。

② 堆栈操作应 PUSH、POP 成对，且 PUSH 先行，保护返回地址。

③ 多重调用应考虑堆栈的容量。

④ 防止不经调用进入子程序，禁止不经返回指令跳出子程序。

（5）单片机应用系统的总体方案设计步骤

▷▷ 22.3 案例设计

▷▷▷ 22.3.1 硬件电路

数字计算器电路如图 22-1 所示。

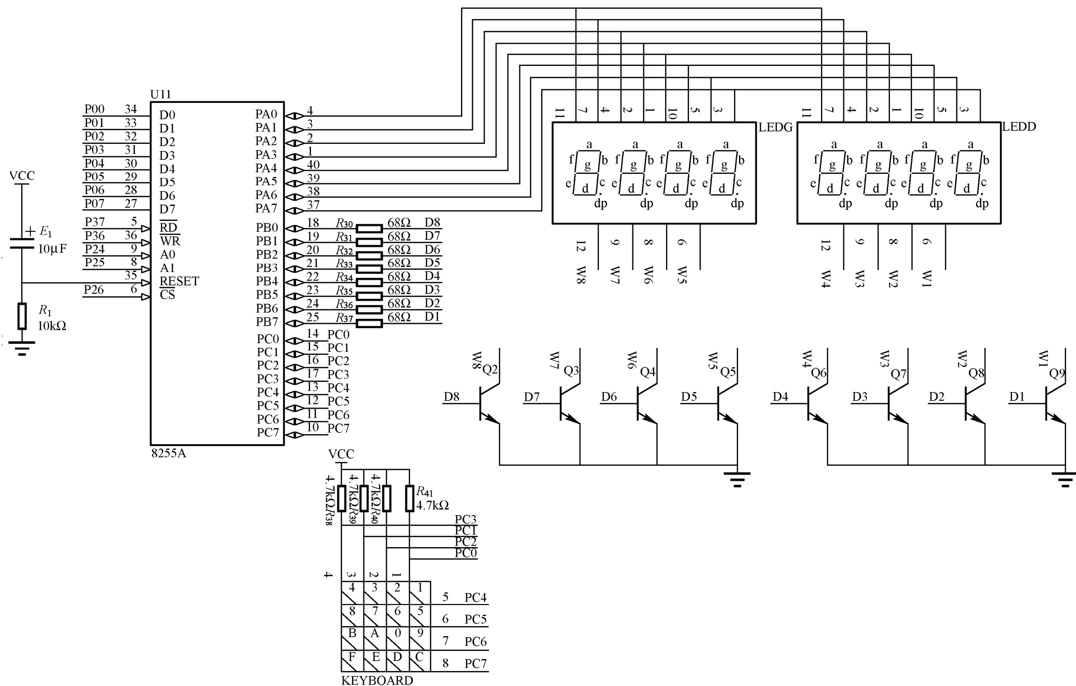


图 22-1 数字计算器电路

22.3.2 软件代码

（1）主程序流程图

主程序流程图如图 22-2 所示。

程序中涉及两个操作数，每个操作数具有四个信息要素，分别是十进制真值、总位数、小数点位数、正负等，因此每个操作数采用结构的存储方式表示：

```
struct operand
    {
        char num_bit;            //操作数总位数
        char num_point;          //操作数小数点后数据位数
        char value[16];          //16 位未压缩 BCD 码表示的操作数
    } operand1,operand2;
```

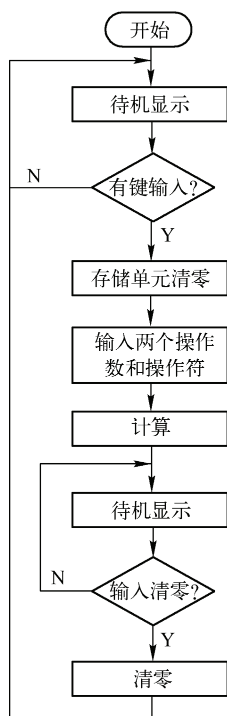


图 22-2 主程序流程图

16 字节的数值单元 value[16]中，最高位存储符号位，若为正数，则存储 0x00；若为负数，则存储 0x0a。键盘输入的值在 operand1 的相应单元中存储；输入操作符加、减、乘、除后，operand1 中的内容传递到 operand2 中，然后 operand1 存储第二个操作数；计算完成后，结果存于 operand1。因此 operand2 保存的是被加数、被减数、被乘数、被除数；operand1 保存的是加数、减数、乘数、除数。由于开方操作只有一个操作数，因此被开放数存于 operand1 中。加、减、乘、除操作符存于 operation 中；结果的符号存于 sign 中。

主函数 main()共有两个 while(1)循环，当有按键输入后进入内部 while(1)循环，其中的函数 keywork()功能为输入两个操作数和操作符 operation。keywork()也有一个 while(1)循环，当输入第一个数时，第一个数的各位满足 if(key>=0&&key<=9)条件，storage_num()将其存入结构体 operand1 中的 value[16]数组；注意 continue 语句，下面的功能操作符输入不再处理，而直接返回 while(1)循环处理下一位输入数字。当输入非 0～9 的数字位，则为功能键输入。

若输入清零 E 键即 key=0x0e，则退出 keywork()子函数并返回 1，主函数内 while(1)经判断返回外 while(1)初始化变量。

若输入第二功能 F 键即 key=0x0f，则 snd 从 0x07 到 0x00 跳转。键盘的复用是通过全局变量 snd 实现的。在键盘处理子函数 keywork()中：

```
if(snd==0x07)
    {
        snd=0;
        P1_7=1;
    }
else
{
        snd=0x07;
        P1_7=0;
}
```

因此当检测到 F 键按下时，snd 从 0x00 到 0x07 进行切换操作。而在键盘扫描子函数中：

```
skey()
if(key==9)                    //数字处理
    key=0;
else
    if(key<=9)                //0～9 为数字
        key++;
if(key>=0x0a&&key<=0x0e)      //a～e 为功能键
   key=key+snd;
```

因此 key 的值为 0x0a～0x0e 时为第一功能；为 0x11～0x15 时为第二功能。

若输入等号键，即 key=0x11，则返回 0 进行 compute()函数运算。

若输入退格键即 key=0x12，则调用 back()函数。

若输入负号键即 key=0x13，首先判断 operand1 的最高位是否为 0x0a（负号），若不是则在最高位置 0x0a，若是负号则负负为正，最高位清零。

若输入为小数点即 key=0x14，则有两种特殊情况需处理：operand1 中无数字；operand1 中仅有负号。

若输入为开根号即 key=0x15，则首先判断 operand1 是否为 0，不为 0 则判断符号位是否为 0x0a（负号），若为 0x0a 则调用 wrong()程序，否则调用 kaigenghao()。注意开根号只有一个操作数，因此 kaigenghao()会直接把结果运算出来。

除了以上输入外，剩下的就是加、减、乘、除运算操作符。输入操作符后，利用 exchange()将 operand1 赋给 operand2，然后返回 while(1)循环接受输入第二个数。当输入第二个数后，按下等号或加减乘除键，则 return 0，回到 main()函数 if(jsf==0x00)条件满足，则调用 compute()函数，运算结果将显示出来。

由于程序中调用子函数较多，因此将子函数分成 BIOS 层、DOS 层和用户程序层，整理情况见表 22-1～表 22-3。

表 22-1　BIOS 层函数

delay()	延时函数
display()	显示子函数
askey()	检查是否有键按下子函数
skey()	键盘扫描子函数，将键值存入 key
back()	退格键子函数

表 22-2　DOS 层函数

storage_num()	存数子函数：将键值存入 operand1
exchange()	交换子函数：operand1 的值全部赋给 operand2，operand1 清零
keywork()	键盘处理子函数：读入键值，并决定下一步的操作

表 22-3 用户程序层函数

max(char a,char b)	求最大值子函数，返回两数中较大的一个
min(char a,char b)	求最小值子函数，返回两数中较小的一个
HEX2BCD()	BCD 转换：将 operand1 转换成 BCD 数，并清零 operand2
uniform_point()	小数点对齐子函数：加减时使小数点对齐
uniform_all()	对齐子函数：除法时数据高位对齐，低位补零
compare(char a,char b)	operand1、operand2 比较大小（比较时不考虑小数点）
rl_1bit()	将 operand2（被除数）左移动一位，用于除法
deal_result(char num_bit,char num_point)	结果处理子函数：包括四舍五入、结果位数超出显示和负号处理
judge_zero(char*operand,num_bit)	判断零子函数：operand 的所有位为零返回 1，否则返回 0
sub_chufa()	减 1 子函数：operand2 减 operand1 一次，用于除法
ADD()	operand1 加 operand2，结果存入 operand1
SUBB()	减法子函数：operand2 减去 operand1，结果存入 operand1
MUL()	乘法子函数：operand1 乘以 operand2，结果存入 operand1
DIV()	除法子函数：operand2 除以 operand1，结果存入 operand1
DIV2()	除 2 子函数：operand1 除以 2 结果存入 operand1
precision(char *s2)	精确子函数：判断开根号是否精确，x[n]=x[n-1]返回 1
kaigenghao()	开根号子函数
wrong()	出错子函数：除数为零时显示 E，清零键复位
compute()	计算子函数：+－×÷运算

（2）uniform_point()子程序分析

uniform_point()子函数是加减法计算的核心内容，本程序单独用一个字节（即操作数结构中的 num_point 变量）保存和小数点后的位数相关的值，该字节保存的值并不是小数位数，而是保存的小数位数加 1。这样做的目的是为了显示的方便。即 num_point 为 0 不显示小数点，num_point 为 1 在最低位显示小数点（注意此时小数位数还是零），num_point 为几就在右边第几个数码管显示小数点。

小数点对齐的目的就是将两个操作数中小数位数较少的数左移 max_point-min_point（参数含义见源程序）位后使两数的小数位数相同，左移后必须补零。

例如，加数为 2.34，被加数为 456，这时存储器的值见表 22-4。

表 22-4 小数点对齐前存储单元内容

operand2.value	234	operand1.value	456
operand2.num_bit	3	operand1.num_bit	3
operand2.num_point	3	operand1.num_point	0

小数点对齐子函数执行后存储器的值见表 22-5。

表 22-5 小数点对齐后存储单元内容

operand2（除数）	234	operand1（被除数）	45600

小数点对齐后 max_point 保存两操作数的小数位数，sum_num_bit 保存两操作数的位数。加减法只要对应位相加减（带进位和借位）就可以了。

（3）除法 DIV()子程序分析

除法子程序的功能是计算 operand2 除以 operand1 并将结果存入 operand1 中。

由于本案例的数据存储采用未压缩 BCD 的方法，所以除法实现起来比较困难。本案例采用类似于竖式的算法。这种方法先将被除数和除数的高位对其，然后从被除数的高位取出和除数相同位数的数，比较这个数和除数的大小，这个数大就用它减去除数并且商的最高位（开始为 0）加 1，直到这个数小于除数。然后从被除数中取出下一位补到该数的后边用同样的方法求取商的次高位。循环上面的操作直到除尽或者商的位数超出 8 位（也就是超出显示）。然后用处理函数处理结果以便于显示。

例如，被除数是 532.4，除数是 0.45，那么存储器的内容见表 22-6。

表 22-6　高位对齐前存储单元内容

operand2.value	5324	operand1.value	45
operand2.num_bit	4	operand1.num_bit	3
operand2.num_point	2	operand1.num_point	3

程序中 operand2.num_point 保存的值是 operand2 的小数位数加 1，程序中 operand1.num_point 保存的值是 operand1 的小数位数加 1。小数点处理就是让 operand2.num_point 的值与 operand2 的小数位数相同， DIV()中的指令

```
if(operand1.num_point>0)
  operand1.num_point--;          //求出 operand1 小数点位数
if(operand2.num_point>0)
  operand2.num_point--;          //求出 operand2 小数点位数
```

执行完毕后，即 operand2.num_point=1，operand1.num_point=2。清除最高位的零使 operand1.num_bit 的值变为 2，高位对齐 uniform_all()子函数执行完毕后存储器的内容见表 22-7。

表 22-7　高位对齐后存储单元内容

operand2.value	5324	operand1.value	4500
operand2.num_bit	4	operand1.num_bit	4
operand2.num_point	1	operand1.num_point	4

compare(operand2.num_bit,operand1.num_bit)把 operand2 和 operand1 的值当作整数比较，本例中 operand2 为 5324，operand1 为 4500，返回 0 则说明 operand1 大，就可以调用 sub_chufa ()子函数。返回值不为零就需要调用 rl_1bit()移位子函数改变被除数。

rl_1bit()子函数的功能是使 operand1 乘以 10，本例调用 rl_1bit()子函数后 operand1 的值变为{0，…，0，4，5，0，0，0}，num_operand1=5。

sub_chufa 子函数的功能是 operand1= operand1－operand2。本例中调用 dec_chufa 子函数后 operand1 的值变为{0，…，0，3，9，6，7，6}，num_operand1=5。



商的小数位数计算公式如下：

$$N=X-Y+Z-1$$

式中，N 为商的小数位数；X 为商的位数；Y 为被除数的小数位数；Z 为除数的小数位数。

（4）减法 SUBB()子程序分析

SUBB()子程序首先进行小数点对齐，通过 uniform_point()子程序实现。然后通过 compare(sum_num_bit,sum_num_bit)判断 operand1 和 operand2 值的大小。由于 uniform_point()已实现小数点对齐，因此在 compare(sum_num_bit,sum_num_bit)子程序中不必考虑小数点。compare(sum_num_bit,sum_num_bit)的比较结果用 fh 记录，若 operand1 大，则 fh=7；若 operand2 大，则 fh=8。

然后从低位开始进行数值相减操作。首先假设有借位发生：

```
operand1.value[i]=operand2.value[i]+22-operand1.value[i];
```

上条语句+10 即表示借位，然后通过 if(operand1.value[i]>9)判断结果是否大于 9，若大于 9，则说明无借位发生，通过

```
operand1.value[i]=operand1.value[i]-10;
```

把借位减掉。若有借位发生，则高位需减去借位，通过

```
operand2.value[i+1]--;
```

实现。

最后处理结果符号位。若 if(fh==sign)，则有两种情况，第一种，sign=8，说明+（operand2-operand1）情况，fh=8，说明 operand2 大，则结果为正即 sign=8；第二种，sign=7，说明-（operand2-operand1）情况，fh=7，说明 operand1 大，则负负得正结果为正即 sign=8。

若 fh 不等于 sign，也有两种情况。第一种，sign=8，说明+（operand2-operand1）情况，fh=7，说明 operand1 大，则结果为负，即 sign=7；第二种，sign=7，说明-（operand2-operand1）情况，fh=8，说明 operand2 大，则结果为负，即 sign=7。

（5）开根号 kaigenghao()子程序分析

开根号子程序有两种实现方案：牛顿迭代法和二分法。采用牛顿迭代法能够减少计算量从而提高程序的效率。所以本案例采用牛顿迭代法，下面详细介绍一下牛顿迭代法。

解方程（代数方程）是最常见的数学问题之一，也是众多应用领域中不可避免的问题之一。目前还没有一般的解析方法来求解非线性方程，但如果在任意给定的精度下，能够解出方程的近似解，则可以认为求解问题已基本解决，至少可以满足实际需要。

设 $f(x)=0$，如果 $f(x)$是一次多项式，则称此方程为线性方程；否则称之为非线性方程。

牛顿迭代法的基本思想为用线性方程来近似非线性方程，即采用线性化方法。设非线性方程 $f(x)=0$，$f(x)$在 x_0 处的 Taylor 展开为

$$f(x)=f(x_0)+f'(x_0)(x-x_0)+\frac{f''(\eta)}{2!}(x-x_0)^2\approx f(x_0)+f'(x_0)(x-x_0)=P(x)$$

令 $P(x)=0$，则

$$x = x_0 - \frac{f(x_0)}{f'(x_0)} \quad f'(x_0) \neq 0$$

牛顿迭代公式

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

牛顿迭代法的几何意义如图 22−3 所示。

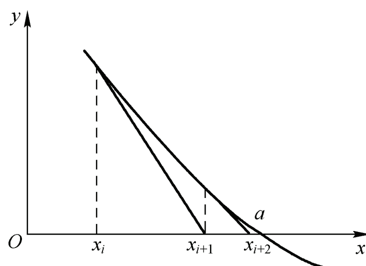


图 22−3　牛顿迭代法的几何意义

由式 $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$ 知 x_{k+1} 是点 $(x_k, f(x_k))$ 处 $y = f(x)$ 的切线 $\frac{y - f(x_k)}{x - x_k} = f'(x_k)$ 与 x 轴的交点的横坐标。也就是说，新的近似值 x_{k+1} 是用代替曲线 $y=f(x)$的切线与 x 轴相交得到的。继续取点 $(x_{k+1}, f(x_{k+1}))$，再做切线与 x 轴相交，又可得 $x_{k+2}, \cdots$。由图可见，只要初值取得充分靠近目标解，这个序列就会很快收敛于目标解。

牛顿迭代法的优点为至少二阶局部收敛，收敛速度较快，特别是当迭代点充分靠近精确解时。

牛顿迭代法的缺点为对重根收敛速度较慢（线性收敛）；对初值的选取很敏感，要求初值相当接近真解。

在实际计算中，可以先用其他方法获得真解的一个粗糙近似，然后再用牛顿迭代法求解。

用牛顿迭代开根号只要设 $f(x)= x^2 - y$，其中 y 为要开根号的数即为一个已知的常数，迭代的结果 x_{k+1} 即为开根号的结果。n 的大小与开根号的精度有关，精度越高 n 越大。

所以牛顿迭代开根号用到的公式为

$$x_{k+1} = x_k - \frac{f(x_k)}{2x_k}$$

初始条件 x_0 设为 2。运算的结束通过精确子函数 precision(char *s2)判断。precision(char *s2)判断 x_{k+1} 和 x_k 是否相等，若相等则说明在有限精度内达到要求，否则继续牛顿迭代开根号。

由于程序中包括加、减、乘、除的运算，所以牛顿迭代法用到的加法和除法不用重新编写子程序，只要把入口条件写好并且知道出口条件就可以直接调用相应的子函数。加、减、乘、除四个子函数的入口条件是第一个操作数（加数、被减数、乘数或被除数）存入 operand1，第二个操作数（被加数、减数、被乘数或除数）存入 operand2。出口条件是结果保存到 operand2。考虑到除以 2 有其特殊性，又新加一个 DIV2()子函数，这样可以提高计算效率。其原因是除法子函数处理的问题远远多于 DIV2()子函数处理的问题。

（6）deal_result(char num_bit,char num_point)子程序分析

deal_result(char num_bit,char num_point)为结果处理子函数，它的主要功能为：

1）处理最高位没用的 0。

2）超出显示位数作四舍五入处理。注意 sign 即表示正负也表示显示位数，sign=8 时为正数，显示位数为 8 位；sign=7 时为负数，显示位数为 7 位。

3）处理最低位没用的 0。

4）处理最高位负号情况。

（7）compute()子程序分析

计算子程序 compute()首先处理负号位，它将 operand1 和 operand2 的正负情况用 sign 表示出来，见表 22-8。



表 22-8　sign 与操作数符号关系

sign	operand1 符号	operand2 符号
8	+	+
5	+	−
6	−	+
7	−	−

然后进行加、减、乘、除运算。若为加法（if(operation==0x0a)），分两种情况：若 operand1 和 operand2 同号，即 if(sign>=7)，则 operand1 和 operand2 绝对值作加法；若 operand1 和 operand2 异号，则 operand1 和 operand2 绝对值作减法。

若为减法（if(operation==0x0b)），也分两种情况：若 operand1 和 operand2 同号，即 if(sign>=7)，则 operand1 和 operand2 绝对值作减法；若 operand1 和 operand2 异号，则 operand1 和 operand2 绝对值作加法，注意 sign+=2 语句，因为 SUBB()中进行的是 operand2−operand1，所以 sign=6 则 operand2−operand1= operand2+（−operand1）结果符号为正，即 sign=6+2=8；若 sign=5 则 operand1−operand2=−（operand2−operand1）结果符号为负，即 sign=5+2=7。

若为乘法（if(operation==0x0c)），则结果的符号同号（if(sign>=7)）为正，异号为负。

除法的结果符号和乘法同理，但除法要判断除数不能为零，为零则转 wrong()函数。

进行完运算操作后，相应存储单元要清零。

（8）源代码

1）汇编程序如下：

```
/*本程序实现浮点数（包括正负）加、减、乘、除和开根号的功能，精确到满屏显示，#键*/
/*为第二功能键，按一下实现第二功能（LED6 亮），再按一下 led6 灭实现第一功能，数*/
/*字键只有第一功能，ABCDE 键对应第一功能是加、乘、等于、小数点、清零，第二功*/
/*能为减、除、退格、负号、开根号。                                          */
/*key 值含义 0～9：输入数字；0x0a：+；0x0b：-；0x0c：×；0x0d：÷；0x0e：清零；*/
/*0x0f：第二功能；0x12：＝；0x13：退格；0x14：负号；0x15：小数点；0x16：开根号；*/
SUMBIT1         DATA        2EH         ;OPERAND1 数的位数
SUMBIT2         DATA        2FH         ;OPERAND2 数的位数
POINT1          DATA        30H         ;OPERAND1 的小数位数加 1
POINT2          DATA        31H         ;OPERAND2 的小数位数加 1
SUMBIT_TMP1     DATA        32H
SUMBIT_TMP2     DATA        33H
POINTTMP1       DATA        34H         ;中间变量
POINTTMP2       DATA        35H
POINTTMP3       DATA        36H
OPERATION       DATA        37H         ;+－×÷运算操作码
```




```
SIGN        DATA    38H             ;正负符号控制
KEY         DATA    39H             ;定义闹钟
SND         DATA    3AH             ;第二功能键标志，按下 F 后 SND=0x07
TEMP1       DATA    3BH             ;自由变量
TEMP2       DATA    3CH
TEMP3       DATA    3DH
TEMP4       DATA    3EH
OPERAND1    DATA    50H             ;OPERAND1 首地址
OPERAND2    DATA    60H             ;OPERAND2 首地址
OPER_TMP1   DATA    70H             ; C 数首地址（C 数为中间量）
OPER_TMP2   DATA    40H
OPER_TMP3   DATA    48H
PORTA       EQU     8FFFH           ;8255A 口地址
PORTB       EQU     9FFFH           ;8255B 口地址
PORTC       EQU     0AFFFH          ;8255C 口地址
CADDR       EQU     0BFFFH          ;8255 控制字地址
            ORG     0000H
            AJMP    MAIN
            ORG     0100H
MAIN:       MOV     SP,#20H         ;初始化堆栈
            MOV     A,#81H
            MOV     DPTR,#CADDR
            MOVX    @DPTR,A         ;初始化 8255
            CLR     P1.2
            MOV     SND,#00H
            SETB    P1.7            ;初始化指示灯
            SETB    P1.6
            SETB    P1.5
            SETB    P1.4
            MOV     POINT1,#00H
            MOV     R7,#2FH
            MOV     R0,#OPERAND1    ;OPERAND1 指针赋给 R0
LOOP:       MOV     @R0,#00H        ;OPERAND1 赋初值 0
            INC     R0
            DJNZ    R7,LOOP
            MOV     SUMBIT1,#00H    ;OPERAND1 数的位数清零
            MOV     SUMBIT2,#00H    ;OPERAND2 数的位数清零
            MOV     OPERATION,#00H  ;运算操作码清零
            MOV     TEMP1,#00H
            MOV     TEMP2,#00H
START:      MOV     R0,#OPERAND1    ;显示指针指向 OPERAND1
            LCALL   DISPLAY
            LCALL   KSI             ;调用键盘扫描
            JZ      START
            LCALL   KEYI
            MOV     A,KEY           ;获得键码
            CJNE    A,#0EH,FQL      ;不是清零键被按下（非清零）
            JMP     MAIN
FQL:        CJNE    A,#0FH,NXT1     ;第二功能键
            MOV     R1,SND
            CJNE    R1,#07H,NXT12
            MOV     SND,#00H        ;if(SND=07H)SND=00H
            SETB    P1.7
            JMP     START
NXT12:      MOV     SND,#07H        ;if(SND=00H)SND=07H
```



```
            CLR     P1.7
            JMP     START
NXT1:       CJNE    A,#14H,NXT2         ;小数点
            MOV     R1,SUMBIT1
            CJNE    R1,#00H,NXT21
            INC     SUMBIT1             ;小数点个数为 0 则个数增 1,显示用
NXT21:      CJNE    R1,#01H,NXT22       ;总位数=1?
            MOV     R0,OPERAND1         ;存储符号的单元
            CJNE    R0,#0AH,NXT22       ;存储符号的单元存"-"
            MOV     OPERAND1,#00H
            MOV     51H,#0AH            ;先负号再 POINT
            INC     SUMBIT1
NXT22:      MOV     R1,POINT1
            CJNE    R1,#00H,START
            INC     POINT1
            JMP     START
NXT2:       CJNE    A,#13H,NXT3         ;负号
            MOV     A,#OPERAND1
            ADD     A,SUMBIT1
            MOV     R1,A
            DEC     R1
            MOV     A,@R1               ;[OPERAND1+SUMBIT1-1]-->A
            CJNE    A,#0AH,NXT31        ;判断是否为负号
            MOV     A,#00H
            MOV     @R1,A
            DEC     SUMBIT1
            JMP     START
NXT31:      INC     R1
            MOV     @R1,#0AH            ;存储"-"
            INC     SUMBIT1
            JMP     START
NXT3:       CJNE    A,#12H,NXT4         ;退格
            LCALL   BACK
            JMP     START
NXT4:       CJNE    A,#15H,NXTB         ;开根号
            MOV     A,#OPERAND1
            ADD     A,SUMBIT1
            MOV     R1,A
            DEC     R1
            MOV     A,@R1               ;[OPERAND1+SUMBIT1-1]-->A
            CJNE    A,#0AH,NXT32        ;判断是否为负号
            ACALL   ERR                 ;负数不能开根号
            JMP     START
NXT32:      LCALL   KGENGHAO
            JMP     START
NXTB:       CLR     CY
            SUBB    A,#0AH
            JNB     ACC.7,NXT5          ;有操作（加、减、乘、除）键按下
            ACALL   STORAGE
            JMP     START
NXT5:       MOV     A,KEY               ;等于键识别
            CJNE    A,#11H,NXT6
            MOV     R0,OPERATION        ;取+－×÷操作码
            CJNE    R0,#00H,NXT51
            JMP     START
NXT51:      LCALL   COMPUTE
```

```
            MOV     OPERATION,#00H
            SETB    P1.6
            SETB    P1.5
            SETB    P1.4
            JMP     START
NXT6:       MOV     A,OPERATION         ;加、减、乘、除处理
            JZ      NXT61               ;没有按过操作键
            LCALL   COMPUTE
            MOV     OPERATION,#00H      ;计算完成后清操作符
            SETB    P1.6
            SETB    P1.5
            SETB    P1.4
            JMP     START
NXT61:      MOV     OPERATION,KEY       ;判断操作符种类
            MOV     R1,OPERATION
            CJNE    R1,#0AH,NXT62
            SETB    P1.6
            SETB    P1.5
            CLR     P1.4
            JMP     NXT65
NXT62:      CJNE    R1,#0BH,NXT63
            SETB    P1.6
            CLR     P1.5
            SETB    P1.4
            JMP     NXT65
NXT63:      CJNE    R1,#0CH,NXT64
            CLR     P1.6
            SETB    P1.5
            SETB    P1.4
            JMP     NXT65
NXT64:      SETB    P1.6
            CLR     P1.5
            CLR     P1.4
NXT65:      MOV     A,SUMBIT1
            JNZ     NXT66
            INC     SUMBIT1
NXT66:      MOV     R7,SUMBIT1
            MOV     R0,#OPERAND1
            MOV     R1,#OPERAND2
TRANS1TO2:  MOV     A,@R0       ;把 50h 为首地址的数放入 60h 为首地址的地方
            MOV     @R1,A
            MOV     @R0,#00H
            INC     R0
            INC     R1
            DJNZ    R7,TRANS1TO2
            MOV     SUMBIT2,SUMBIT1
            MOV     POINT2,POINT1
            MOV     POINT1,#00H
            MOV     SUMBIT1,#00H
            JMP     START
;以下为子函数
BACK:       MOV     A,SUMBIT1           ;退格处理子函数
            JZ      ENDBACK
            DEC     A
            JZ      LP
            MOV     R7,SUMBIT1
            DEC     R7
            MOV     R1,#OPERAND1
```



```
            MOV     R0,#OPERAND1
            INC     R1                  ;OPERAND1[i-1]= OPERAND1[i]
TGLOOP:     MOV     A,@R1
            MOV     @R0,A
            INC     R1
            INC     R0
            DJNZ    R7,TGLOOP
            MOV     @R0,#00H
            DEC     SUMBIT1
            MOV     A,POINT1
            JZ      ENDBACK
            DEC     POINT1
ENDBACK:    RET
LP:         MOV     A,POINT1            ;小数点处理
            JZ      TGW
            MOV     POINT1,#00H
            RET
TGW:        DEC     SUMBIT1
            MOV     OPERAND1,#00H
            RET
STORAGE:    MOV     A,SUMBIT1           ;计数子函数（读入数据）
            CJNE    A,#08H,WYCHU        ;超过 8 位时不记录，最多能显示 8 位
            RET
WYCHU:      MOV     A,KEY               ;接受键盘数字并且保存
            CJNE    A,#00H,OK           ;第一个输入的数据是零不记录
            MOV     R0,SUMBIT1
            CJNE    R0,#00H,OK          ;第一个输入的数据是零不记录
            RET
OK:         MOV     A,#OPERAND1
            ADD     A,SUMBIT1
            MOV     R0,A
            MOV     R1,A
            DEC     R1
            MOV     A,SUMBIT1
            JZ      XYG                 ;如果是输入第一个数则直接记录
            MOV     R7,SUMBIT1
MM:         MOV     A,@R1               ;不是第一个数要移动数据后再记录
            MOV     @R0,A
            DEC     R1
            DEC     R0
            DJNZ    R7,MM
XYG:        MOV     @R0,KEY
            INC     SUMBIT1
            MOV     A,POINT1            ;是小数则小数点加 1
            JZ      STOREND
            INC     POINT1
STOREND:    RET
COMPUTE:    MOV     A,#OPERAND1         ;运算子函数
            ADD     A,SUMBIT1
            DEC     A
            MOV     R0,A                ;取出 OPERAND1 符号位
            MOV     A,#OPERAND2
            ADD     A,SUMBIT2
            DEC     A
            MOV     R1,A                ;取出 OPERAND2 符号位
            MOV     SIGN,#00H           ;处理符号（正负）
```

```
        MOV     A,@R0
        CJNE    A,#0AH,JIS1
        INC     SIGN                ;是负数则用符号记录
        MOV     A,#00H              ;并清除最高位的 0AH
        MOV     @R0,A
        DEC     SUMBIT1
JIS1:   MOV     A,@R1
        CJNE    A,#0AH,JIS2
        INC     SIGN
        INC     SIGN
        MOV     A,#00H
        MOV     @R1,A
        DEC     SUMBIT2
JIS2:   MOV     R1,OPERATION
        CJNE    R1,#0AH,NJIA        ;加法处理
        MOV     A,SIGN
        JNZ     NJIA1               ;SIGN 为 0 说明正、正相加
        MOV     SIGN,#08H
        LCALL   ADDT
        JMP     CMP_END
NJIA1:  DEC     A
        JNZ     NJIA2               ;SIGN 为 1 说明负、正相加
        MOV     SIGN,#08H
        LCALL   SUBT
        JMP     CMP_END
NJIA2:  DEC     A
        JNZ     NJIA3               ;SIGN 为 2 说明正、负相加
        MOV     SIGN,#07H
        LCALL   SUBT
        JMP     CMP_END
NJIA3:  MOV     SIGN,#07H           ;SIGN 为 3 说明负、负相加
        LCALL   ADDT
        JMP     CMP_END
NJIA:   CJNE    R1,#0BH,NJIE        ;减法处理
        MOV     A,SIGN
        JNZ     NJIE1               ;SIGN 为 0 说明正、正相减
        MOV     SIGN,#08H
        LCALL   SUBT
        JMP     CMP_END
NJIE1:  DEC     A                   ;SIGN 为 1 说明负、正相减
        JNZ     NJIE2
        MOV     SIGN,#08H
        LCALL   ADDT
        JMP     CMP_END
NJIE2:  DEC     A                   ;SIGN 为 2 说明正、负相减
        JNZ     NJIE3
        MOV     SIGN,#07H
        LCALL   ADDT
        JMP     CMP_END
NJIE3:  MOV     SIGN,#07H           ;SIGN 为 3 说明负、负相减
        LCALL   SUBT
        JMP     CMP_END
NJIE:   CJNE    R1,#0CH,NCHENG      ;乘法处理
        LCALL   JUDGE_SIGN
        MOV     R0,#OPERAND1
        MOV     R7,SUMBIT1
        LCALL   JUDGE_0             ;乘数是否为零
```



```
            JZ      CSHU0
            MOV     R0,#OPERAND2
            MOV     R7,SUMBIT2
            LCALL   JUDGE_0             ;被乘数是否为零
            JZ      CSHU0
            LCALL   MULT
            JMP     CMP_END
CSHU0:      MOV     SUMBIT1,#00H
            JMP     CMP_END
NCHENG:     CJNE    R1,#0DH,CMP_END
            MOV     R0,#OPERAND1        ;除法处理
            MOV     R7,SUMBIT1
            LCALL   JUDGE_0
            JNZ     NERR
            ACALL   ERR                 ;除数为零出错
            RET
NERR :      MOV     R0,#OPERAND2
            MOV     R7,SUMBIT2
            LCALL   JUDGE_0
            JZ      CSHU0               ;被除数为零结果为零
            LCALL   JUDGE_SIGN
            LCALL   DIVT
            JMP     CMP_END
CMP_END:    MOV     A,#2FH
            CLR     CY
            SUBB    A,SUMBIT1
            MOV     R7,A                ;无关存储器清零
            MOV     A,#OPERAND1
            ADD     A,SUMBIT1
            MOV     R0,A
JSLOOP:     MOV     @R0,#00H
            INC     R0
            DJNZ    R7,JSLOOP
            MOV     SUMBIT2,#00H
            RET
ERR:        MOV     SUMBIT1,#00H        ;出错显示
            MOV     OPERAND1,#0BH
            LCALL   DISPLAY
            MOV     SND,#00H            ;清 SND
            SETB    P1.7
            LCALL   KSI
            JZ      ERR
            LCALL   KEYI
            MOV     R1,KEY
            CJNE    R1,#0EH,ERR         ;清零键复位
            MOV     OPERAND1,#00H
            RET
DELAY:      MOV     R6,#0FFH            ;延时子函数
DLOOP:      NOP
            DJNZ    R6,DLOOP
            DJNZ    R7,DELAY
            RET
KGENGHAO:   MOV     R7,SUMBIT1          ;用减法求商
            MOV     R0,#OPERAND1
            LCALL   JUDGE_0
            JNZ     KGH0
            MOV     SUMBIT1,#00H
```




```
        MOV     POINT1,#00H
        RET
KGH0:   MOV     SIGN,#08H
        MOV     R1,POINT1
        MOV     R0,#OPER_TMP2
        MOV     R7,#10H
        CJNE    R1,#01H,KGH1            ;小数点 1 等于 1 说明没有小数位
        MOV     POINT1,#00H
KGH1:   MOV     A,#00H                  ;OPER_TMP2 清零
        MOV     @R0,A
        INC     R0
        DJNZ    R7,KGH1
        MOV     R0,#OPER_TMP2
        MOV     R1,#OPERAND1
        MOV     R7,SUMBIT1
        LCALL   TRANSF          ;Y[n-1](OPERAND1)清零保存到 OPER_TMP2
        MOV     SUMBIT_TMP1,SUMBIT1
        MOV     POINTTMP2,POINT1
        MOV     OPER_TMP3,#02H                  ;x[0]=2
        MOV     SUMBIT_TMP2,#01H                ;x[0]总位数
        MOV     POINTTMP3,#00H                  ;x[0]小数位数
        MOV     R4,#20H
KGH2:   MOV     KEY,R4
        MOV     R7,SUMBIT_TMP2
        MOV     R0,#OPERAND1
        MOV     R1,#OPER_TMP3
        LCALL   TRANSF                          ;x[n-1]-->OPERAND1
        MOV     POINT1,POINTTMP3
        MOV     SUMBIT1,SUMBIT_TMP2
        MOV     R0,#OPERAND1
        MOV     R7,SUMBIT1
        LCALL   CLEAN_0
        MOV     R7,SUMBIT_TMP1
        MOV     R0,#OPERAND2
        MOV     R1,#OPER_TMP2
        LCALL   TRANSF                          ;y[n-1]-->OPERAND2
        MOV     POINT2,POINTTMP2
        MOV     SUMBIT2,SUMBIT_TMP1
        MOV     R0,#OPERAND2
        MOV     R7,SUMBIT2
        LCALL   CLEAN_0
        MOV     R0,#OPER_TMP1
        MOV     R7,#00H
        LCALL   CLEAN_0
        LCALL   DIVT                            ;y[n-1]/x[n-1]
        MOV     R0,#OPERAND1
        MOV     R7,SUMBIT1
        LCALL   CLEAN_0
        MOV     R7,SUMBIT_TMP2
        MOV     R0,#OPERAND2
        MOV     R1,#OPER_TMP3
        LCALL   TRANSF                          ;x[n-1]-->OPERAND2
        MOV     POINT2,POINTTMP3
        MOV     SUMBIT2,SUMBIT_TMP2
        MOV     R0,#OPERAND2
        MOV     R7,SUMBIT2
        LCALL   CLEAN_0
```



```
                LCALL   ADDT                    ;y[n−1]/x[n−1]+x[n−1]
                MOV     R7,SUMBIT1
                MOV     R0,#OPERAND2
                MOV     R1,#OPERAND1
                LCALL   TRANSF                  ;OPERAND1-->OPERAND2
                MOV     POINT2,POINT1
                MOV     SUMBIT2,SUMBIT1
                MOV     R0,#OPERAND2
                MOV     R7,SUMBIT2
                LCALL   CLEAN_0
                MOV     SUMBIT1,#01H
                MOV     OPERAND1,#02H
                MOV     POINT1,#00H
                MOV     R0,#OPERAND1
                MOV     R7,SUMBIT1
                LCALL   CLEAN_0
                MOV     R0,#OPER_TMP1
                MOV     R7,#00H
                LCALL   CLEAN_0
                LCALL   DIVT                    ;(y[n−1]/x[n−1]+x[n−1])/2
                LCALL   PRECISION
                JZ      KGH3
                MOV     R0,#OPER_TMP3
                MOV     R1,#OPERAND1
                MOV     R7,SUMBIT1
                LCALL   TRANSF                  ;x[n]-->OPER_TMP3
                MOV     SUMBIT_TMP2,SUMBIT1
                MOV     POINTTMP3,POINT1
KH5:            JMP     KGH2
KGH3:           MOV     R0,#OPERAND2
                MOV     R7,SUMBIT2
                LCALL   CLEAN_0
                MOV     R0,#OPER_TMP1
                MOV     R7,#00H
                LCALL   CLEAN_0
                RET
;x[n]=x[n-1]认为开根号结果正确
PRECISION:      MOV     R0,#OPERAND1
                MOV     R7,SUMBIT1
                LCALL   CLEAN_0
                MOV     A,SUMBIT1
                CJNE    A,SUMBIT_TMP2,JQUE2     ;位数不等则认为不精确
                MOV     A,POINT1
                CJNE    A,POINTTMP3,JQUE2       ;小数位数不等则认为不精确
                MOV     R7,SUMBIT1
                MOV     R0,#OPERAND1
                MOV     R1,#OPER_TMP3
JQUE1:          MOV     A,@R0
                CLR     CY
                SUBB    A,@R1
                JNZ     JQUE2                   ;有一位不等则认为不精确
                INC     R0
                INC     R1
                DJNZ    R7,JQUE1
                MOV     A,#00H                  ;精确返回0
                RET
JQUE2:          MOV     A,#01H                  ;不精确返回1
```

```
                RET
; 传数子函数：将以 R1 为首地址的数传给以 R2 为首地址的数，传数的位数保存在 R7 中
TRANSF:         MOV     A,@R1
                MOV     @R0,A
                INC     R0
                INC     R1
                DJNZ    R7,TRANSF
                RET
CLEAN_0:        MOV     A,R0
                ADD     A,R7
                MOV     R0,A
                MOV     A,#10H
                CLR     CY
                SUBB    A,R7
                MOV     R7,A
QLLOOP:         MOV     A,#00H
                MOV     @R0,A
                INC     R0
                DJNZ    R7,QLLOOP
                RET
JUDGE_SIGN:     MOV     A,SIGN                  ;乘除用到的处理正负的子函数
                JNZ     CHENG1
                MOV     SIGN,#08H
                RET
CHENG1:         CJNE    A,#03H,CHENG2
                MOV     SIGN,#08H
                RET
CHENG2:         MOV     SIGN,#07H
                RET
JUDGE_0:        MOV     A,R7                    ;判断数是否为零（除法用）
                JNZ     PD1
                RET
PD1:            MOV     A,@R0
                JZ      PD2
                RET
PD2:            INC     R0
                DJNZ    R7,PD1
                RET
ADDT:           LCALL   UNIFORM_POINT           ;加法子函数
                MOV     R7,SUMBIT1
                MOV     R0,#OPERAND1
                MOV     R1,#OPERAND2
                CLR     CY
LOOPJ:          MOV     A,@R1
                ADDC    A,@R0
                DA      A
                JNB     ACC.4,JFB               ;由 ACC.4 决定是否有进位
                SETB    CY
                JMP     JFC
JFB:            CLR     CY
JFC:            CLR     ACC.4
                MOV     @R0,A
                MOV     A,#00H
                MOV     @R1,A
                INC     R1
                INC     R0
                DJNZ    R7,LOOPJ
```



```
            JNC     JFA
            MOV     @R0,#01H            ;最高位有进位，数的位数要加 1
            INC     SUMBIT1
JFA:        LCALL   DEAL_RESULT
            RET
SUBT:       LCALL   UNIFORM_POINT       ;减法子函数
            LCALL   COMPARE             ;OPERAND2 大返回 0
            JZ      JIEF2               ;处理符号，并且决定被减数和减数
            INC     SIGN
            MOV     R1,SIGN
            CJNE    R1,#09H,JIEF1
            MOV     SIGN,#07H
JIEF1:      MOV     R1,#OPERAND1        ;OPERAND1 为被减数
            MOV     R0,#OPERAND2
            JMP     JIEF3
JIEF2:      MOV     R0,#OPERAND1
            MOV     R1,#OPERAND2
JIEF3:      MOV     R7,SUMBIT1
            CLR     CY
JIEF4:      MOV     A,@R1
            SUBB    A,@R0
            MOV     @R1,A
            JNB     ACC.7,BMJ           ;借位处理
            ADD     A,#0AH
            MOV     @R1,A
BMJ:        MOV     @R0,#00H
            INC     R0
            INC     R1
            DJNZ    R7,JIEF4
            MOV     R7,SUMBIT1
            MOV     A,#OPERAND1
            ADD     A,SUMBIT1
            DEC     A
            MOV     R0,A                ;求出 OPERAND1 的最高位
            DEC     R1                  ;R1 存的是结果的最高位
JIEF5:      MOV     A,@R1               ;结果存入 OPERAND1 用于显示
            MOV     @R0,A
            DEC     R0
            DEC     R1
            DJNZ    R7,JIEF5
            LCALL   DEAL_RESULT
            RET
;乘法子函数：实现 OPERAND1=OPERAND1*OPERAND2（竖式原理）
MULT:       MOV     TEMP3,#00H          ;乘法子函数
            MOV     R0,#OPERAND1
            MOV     R1,#OPERAND2
            MOV     R2,SUMBIT1
            MOV     R3,SUMBIT2
            MOV     TEMP4,#00H
CHEF1:      MOV     TEMP3,TEMP4
CHEF2:      MOV     A,@R0
            MOV     B,A
            MOV     A,@R1
            MUL     AB
            MOV     B,#0AH
            DIV     AB
            MOV     TEMP2,A             ;保存积的进位
```

```
            MOV     TEMP1,R0            ;保存 OPERAND1 的位地址
            MOV     A,#OPER_TMP1
            ADD     A,TEMP3
            MOV     R0,A                ;求出积的存储位保存在 R0
            MOV     A,B
            ADD     A,@R0
            MOV     @R0,A               ;将积加到积的存储位
            INC     R0
            MOV     A,TEMP2
            ADD     A,@R0
            MOV     @R0,A               ;将积的进位存入积的进位存储位
            INC     TEMP3               ;改变积的存储位
            INC     R1
            MOV     R0,TEMP1            ;取出 OPERAND1 的位地址
            DJNZ    R3,CHEF2
            INC     R0                  ;改变被乘数位
            INC     TEMP4               ;改变积位
            MOV     R1,#OPERAND2
            MOV     R3,SUMBIT2
            DJNZ    R2,CHEF1
            MOV     A,SUMBIT1           ;进位并存储数据
            ADD     A,SUMBIT2
            MOV     SUMBIT1,A           ;保存积的位数
            MOV     R7,A
            MOV     R0,#OPERAND1
            MOV     R1,#OPER_TMP1
            MOV     TEMP1,#00H
CHEF3:      MOV     A,@R1               ;进位处理
            ADD     A,TEMP1
            MOV     B,#0AH
            DIV     AB
            MOV     TEMP1,A
            MOV     A,B
            MOV     @R0,A
            INC     R1
            INC     R0
            DJNZ    R7,CHEF3
            MOV     A,POINT1
            JZ      CHF4
            DEC     POINT1
CHF4:       MOV     A,POINT2            ;结果小数位数处理
            JZ      CHF5
            DEC     POINT2
CHF5:       MOV     A,POINT1
            ADD     A,POINT2
            MOV     POINT1,A
            ACALL   DEAL_RESULT
            RET
DIVT:       MOV     A,#OPERAND1         ;除法部分
            ADD     A,SUMBIT1           ;处理 OPERAND1 高位没用的零
            DEC     A
            MOV     R0,A
            MOV     A,@R0
            JNZ     CHUF1
            MOV     A,SUMBIT1
            JZ      CHUF1
            DEC     SUMBIT1
```



```
                JMP     DIVT
CHUF1:          MOV     A,#OPERAND2             ;处理 OPERAND2 高位没用的零
                ADD     A,SUMBIT2
                DEC     A
                MOV     R0,A
                MOV     A,@R0
                JNZ     CHUF2
                MOV     A,SUMBIT2
                JZ      CHUF2
                DEC     SUMBIT2
                JMP     CHUF1
CHUF2:          LCALL   POINT_DEC1              ;小数点位数减 1
                MOV     TEMP1,SUMBIT1           ;数字对齐，除法前期准备
                MOV     A,SUMBIT1
                CLR     CY
                SUBB    A,SUMBIT2
                JZ      CHUF5
                JB      ACC.7,CHUF3
                MOV     R7,SUMBIT2  ;OPERAND1 位数多于 OPERAND2，移动 OPERAND2
                MOV     POINTTMP1,POINT2
                MOV     R0,#OPERAND2
                MOV     TEMP1,SUMBIT1
                JMP     CHUF4
CHUF3:          MOV     R7,SUMBIT1  ;OPERAND1 位数少于 OPERAND2，移动 OPERAND1
                MOV     R0,#OPERAND1
                MOV     POINTTMP1,POINT1
                MOV     TEMP1,SUMBIT2
CHUF4:          MOV     SUMBIT1,TEMP1
                MOV     SUMBIT2,TEMP1
                MOV     TEMP2,R0
                MOV     A,R0
                ADD     A,R7
                DEC     A
                MOV     R0,A                    ;求出要移动的数的最高位存到 R0
                MOV     A,TEMP2
                ADD     A,SUMBIT1
                DEC     A
                MOV     R1,A                    ;求出要移动数最高位要移到的位置
CFLOOP1:        MOV     A,@R0                   ;移动数据
                MOV     @R1,A
                DEC     R0
                DEC     R1
                DJNZ    R7,CFLOOP1
                DEC     TEMP2
CFLOOP2:        MOV     @R1,#00H                ;移动后数据补零
                DEC     R1
                INC     POINTTMP1
                MOV     A,R1
                CJNE    A,TEMP2,CFLOOP2         ;数字对齐结束
                MOV     A,TEMP2
                SUBB    A,#50H
                JB      ACC.7,CHUFF
                MOV     POINT2,POINTTMP1        ;数字对齐后对小数点的改变
                JMP     CHUF5
CHUFF:          MOV     POINT1,POINTTMP1
CHUF5:          MOV     SUMBIT1,TEMP1
                MOV     A,POINT1
```

```
                CLR     CY
                SUBB    A,POINT2
                JB      ACC.7,CHUF6     ;记录小数点的位置，以便求取商小数点的位置
                MOV     POINTTMP1,A     ;POINTTMP1 记录(POINT1-POINT2)的绝对值
                MOV     TEMP1,#01H              ;TEMP1=1 表示 POINT1 大
                JMP     CHUF
CHUF6:          CPL     A
                INC     A
                MOV     POINTTMP1,A
                MOV     TEMP1,#00H              ;TEMP1=0 表示 POINT2 大
CHUF:           MOV     R6,#09H                 ;除法前存储器赋值（循环次数）
                MOV     TEMP3,#00H              ;存储本次循环求出的商的位置
CFLOOP3:        MOV     R7,SUMBIT1              ;用减法求商
                MOV     R0,#OPERAND2
                LCALL   JUDGE_0
                JNZ     CHUF7
                JMP     CHUF9                   ;除尽后退出循环
CHUF7:          LCALL   COMPARE
                JNZ     CHUF8
                LCALL   SUB_DIV                 ;被除数减 1 次除数
                MOV     A,#OPER_TMP1
                ADD     A,TEMP3
                MOV     R1,A
                INC     @R1                     ;商的该位加 1
                JMP     CHUF7
CHUF8:          LCALL   RL_1BIT                 ;移位，求商的下 1 位，此处由高到
                                                ;低逆序存储
                INC     TEMP3
                DJNZ    R6,CFLOOP3              ;结束
                INC     TEMP3
CHUF9:          MOV     A,TEMP3                 ;根据 TEMP3 判断循环次数
                JNZ     CHUF9_1
                MOV     SUMBIT1,#00H
                MOV     50H,#00H
                RET
CHUF9_1:        MOV     A,TEMP1                 ;处理结果的小数点位置和结果位数
                JZ      CHUF11
                MOV     A,TEMP3
                SETB    CY
                                                ;根据 TEMP3 和 POINTTMP1 计算结
                SUBB    A,POINTTMP1
                                                ;果小数点的位置
                JB      ACC.7,CHUF10
                MOV     SUMBIT1,TEMP3
                MOV     POINT1,A
                MOV     R4,SUMBIT1
                JMP     CHUF12
CHUF10:         MOV     A,POINTTMP1
                INC     A
                MOV     SUMBIT1,A
                MOV     POINT1,#00H
                MOV     R4,SUMBIT1
                JMP     CHUF12
CHUF11:         MOV     A,TEMP3
                ADD     A,POINTTMP1
                DEC     A
                MOV     POINT1,A
                INC     A
```



```
                MOV     SUMBIT1,A
                MOV     R4,TEMP3
CHUFAX:         MOV     R5,TEMP3    ;把结果从中间寄存器存到 OPERAND1 中用于显示
                MOV     R1,#OPER_TMP1
                DEC     R4
                MOV     A,#OPERAND1
                ADD     A,R4
                MOV     R0,A
CHUFAY:         MOV     A,@R1
                MOV     @R0,A
                DEC     R0
                INC     R1
                DJNZ    R5,CHUFAY                ;结束
                MOV     A,TEMP3
                ADD     A,#OPERAND1
                MOV     R0,A
CHUFAZ:         MOV     @R0,#00H
                INC     R0
                CJNE    R0,#5FH,CHUFAZ
                JMP     CHUF14
CHUF12:         MOV     R5,SUMBIT1  ;结果从中间寄存器存到 OPERAND1 中用于显示
                MOV     R1,#OPER_TMP1
                DEC     R4
                MOV     A,#OPERAND1
                ADD     A,R4
                MOV     R0,A
CHUF13:         MOV     A,@R1
                MOV     @R0,A
                DEC     R0
                INC     R1
                DEC     R4
                DJNZ    R5,CHUF13
CFLOOP4:        CJNE    R0,#4FH,CHUF15
                JMP     CHUF14
CHUF15:         MOV     A,#00H              ;低位补零
                MOV     @R0,A
                DEC     R0
                JMP     CFLOOP4             ;结束
CHUF14:         LCALL   DEAL_RESULT         ;处理结果（说明见函数部分）
                RET
SUB_DIV:        MOV     R0,#OPERAND1        ;OPERAND2 减 OPERAND1 一次
                MOV     R1,#OPERAND2
                CLR     CY
                MOV     R7,SUMBIT1
CJLOOP:         MOV     A,@R1
                SUBB    A,@R0
                MOV     @R1,A
                JNB     ACC.7,CFBMJ
                ADD     A,#0AH
                MOV     @R1,A
CFBMJ:          INC     R0
                INC     R1
                DJNZ    R7,CJLOOP
                RET
RL_1BIT:        MOV     A,#OPERAND2         ;不够减时 BSHU 左移 1 位
                ADD     A,SUMBIT1
                MOV     R0,A
                MOV     R1,A
```




```
                MOV     TEMP4,R1
                DEC     R0
    YWLOOP:     MOV     A,@R0
                MOV     @R1,A
                DEC     R0
                DEC     R1
                CJNE    R1,#OPERAND2,YWLOOP
                MOV     @R1,#00H
                MOV     R0,TEMP4
                MOV     A,@R0
                JZ      YW1
                INC     SUMBIT1                 ;最高位不为零，数的位数加 1
    YW1:        RET
    /*  小数点处理子函数：使小数点 1，2 中存的数变为小数位    */
    /*   数（原来存的是小数位数加一），如果是 0 说明是整数    */
    POINT_DEC1: MOV     A,POINT1                ;小数点处理子函数
                JZ      JF1
                DEC     POINT1
    JF1:        MOV     A,POINT2
                JZ      JF2
                DEC     POINT2
    JF2:        RET
    COMPARE:    MOV     A,#OPERAND1             ;比较两数大小
                ADD     A,SUMBIT1
                DEC     A
                MOV     R0,A
                MOV     A,#OPERAND2
                ADD     A,SUMBIT1
                DEC     A
                MOV     R1,A
                MOV     R7,SUMBIT1
    CMP3:       MOV     A,@R1
                CLR     CY
                SUBB    A,@R0
                JNB     ACC.7,CMP1
                MOV     A,#01H
                RET
    CMP1:       JNZ     CMP2
                DEC     R1
                DEC     R0
                DJNZ    R7,CMP3
    CMP2:       MOV     A,#00H
                RET
/*   小数点对齐子函数：加减时使小数点对齐（使 OPERAND1        */
/*   和 OPERAND2 的小数位数相同，小数位数少的低位补零）       */
UNIFORM_POINT: LCALL  POINT_DEC1               ;小数点对齐子函数
                MOV     POINTTMP1,POINT1
                MOV     A,POINT1
                CLR     CY
                SUBB    A,POINT2            ;OPERAND1 和 OPERAND2 哪个小
                                            ;数位数多
                JZ      JF7                 ;小数位数相等不处理
                JB      ACC.7,JF3
                MOV     POINTTMP1,POINT1    ;POINT1 大，移动 OPERAND2
                MOV     R3,A                ;要移动的位数存入 R3
                MOV     R0,#OPERAND2        ;要移动的数的首地址存入 R0
```



```
            MOV     R4,SUMBIT2              ;要移动数的个数存入 R4
            JMP     JF4
JF3:        MOV     POINTTMP1,POINT2        ;POINT2 大，移动 OPERAND1
            MOV     A,POINT2
            CLR     CY
            SUBB    A,POINT1
            MOV     R3,A                    ;要移动的位数存入 R3
            MOV     R0,#OPERAND1            ;要移动的数的首地址存入 R0
            MOV     R4,SUMBIT1              ;要移动数的个数存入 R4
JF4:        MOV     TEMP1,R0
            MOV     A,R0
            ADD     A,R4
            DEC     A
            MOV     R0,A                    ;要移动数的最高位存入 R0
            ADD     A,R3
            MOV     R1,A                    ;要移动的数的目标地址存入 R1
JLOOP:      MOV     A,@R0                   ;移动数据
            MOV     @R1,A
            DEC     R0
            DEC     R1
            DJNZ    R4,JLOOP
            MOV     R0,T EMP1
JF8:        MOV     A,#00H                  ;低位补零
            MOV     @R0,A
            INC     R0
            DJNZ    R3,JF8
JF7:        MOV     A,SUMBIT1
            CLR     CY
            SUBB    A,POINT1
            MOV     R5,A                    ;OPERAND1 的整数位数存入 R5
            MOV     A,SUMBIT2
            CLR     CY
            SUBB    A,POINT2
            MOV     R6,A                    ;OPERAND2 的整数位数存入 R6
            SUBB    A,R5
            JB      ACC.7,JF5               ;OPERAND1 的整数位数多，跳到 JF5
            MOV     A,R6
            JMP     JF6
JF5:        MOV     A,R5                    ;整数位数的最大值存入 A
JF6:        ADD     A,POINTTMP1             ;求出移动后数据的位数
            MOV     SUMBIT1,A
            MOV     POINT1,POINTTMP1
            RET
DEAL_RESULT: MOV    A,#OPERAND1             ;结果处理子函数
            ADD     A,SUMBIT1
            DEC     A
            MOV     R0,A                    ;OPERAND1 最高位
            MOV     A,@R0
            JN Z    CH1
            MOV     A,SUMBIT1               ;最高位为 0，总位数减 1
            DEC     A
            SETB    CY
            SUBB    A,POINT1
            JB      ACC.7,CH1               ;判断 AGE>POINT1
            DEC     SUMBIT1
            JMP     DEAL_RESULT
```

```
CH1:        MOV     R4,SUMBIT1
            MOV     A,SUMBIT1
            SETB    CY
            SUBB    A,SIGN          ;SUMBIT1>SIGN，判断是否超出显示
            JB      ACC.7,CH2       ;没有超出显示转 CH2
            MOV     R5,A            ;超出显示的位数存入 R5
            INC     R5
            MOV     R4,SIGN
            JMP     CH3
CH2:        MOV     R5,#00H         ;R5 等于 0 说明没有超出显示
CH3:        MOV     A,R5
            JZ      CH5             ;没有超出显示不处理
            ADD     A,#OPERAND1     ;求要四舍五入的位
            DEC     A
            MOV     R0,A            ;要四舍五入的位存入 R0
            MOV     A,@R0
            CLR     CY
            SUBB    A,#05H          ;四舍五入
            JB      ACC.7,CH4
JINWEI:     CLR     ACC.4           ;进位处理（循环的第 1 次不起作用）
            MOV     @R0,A
            INC     R0
            INC     @R0             ;五入
            MOV     A,@R0
            DA      A
            JB      ACC.4,JINWEI    ;五入后有进位到 JINWEI
            MOV     A,#OPERAND1
            ADD     A,SUMBIT1
            MOV     R0,A
            MOV     A,@R0
            JZ      CH4
            INC     R4              ;进位后最高位不为零数的位数加 1
CH4:        MOV     A,POINT1        ;四舍五入后修正 POINT1
            CLR     CY
            SUBB    A,R5
            MOV     POINT1,A
CH5:        MOV     A,#OPERAND1
            ADD     A,R5
            MOV     R0,A            ;找到四舍五入后的最低位
            MOV     A,@R0
            JNZ     CH6             ;记录最低位对数大小不起作用的
                                    ;零的个数
            MOV     A,POINT1        ;OPERAND1 为整数不处理
            JZ      CH6
            DEC     POINT1
            DEC     R4              ;数的位数减 1
            INC     R5              ;要去掉的位数加 1
            JMP     CH5             ;循环，直到最后 1 位不为零
CH6:        MOV     R1,#OPERAND1
            MOV     TEMP1,R4
CHA:        MOV     A,@R0           ;移动数据（清除没用的位）
            MOV     @R1,A
            INC     R0
            INC     R1
            DJNZ    R4,CHA
            MOV     R4,TEMP1
```



```
        INC     R5
CH7:    MOV     A,#00H          ;高位清零
        MOV     @R1,A
        INC     R1
        DJNZ    R5,CH7
        MOV     SUMBIT1,R4
        MOV     A,R4
        SETB    CY
        SUBB    A,SIGN
        JNB     ACC.7,CH1       ;再次判断是否超出显示
        MOV     A,POINT1
        JZ      CH8
        INC     POINT1          ;修正 POINT1 用于显示
CH8:    MOV     R0,SUMBIT1
        CJNE    R0,#01H,CH9
        MOV     A,OPERAND1      ;只有 1 位且为 0 则 OP1_NUM 清零
        JNZ     CH9
        DEC     SUMBIT1
        MOV     SIGN,#08H
CH9:    MOV     R1,SIGN         ;负号的存储（用 0A 表示）
        CJNE    R1,#07H,CH10
        MOV     A,#OPERAND1
        ADD     A,SUMBIT1
        MOV     R0,A
        MOV     @R0,#0AH
        INC     SUMBIT1
CH10:   RET
KEYI:   MOV     R1,#04H         ;扫描键盘
        MOV     R2,#0EFH
LK1:    MOV     DPTR,#PORTC
        MOV     A,R2
        MOVX    @DPTR,A
        NOP
        NOP
        MOVX    A,@DPTR
        JB      ACC.0,LONE      ;第 1 列没有键按下到 LONE
        MOV     A,#00H
        AJMP    LKP
LONE:   JB      ACC.1,LTWO      ;第 2 列没有键按下到 LTWO
        MOV     A,#04H
        JMP     LKP
LTWO:   JB      ACC.2,LTHR      ;第 3 列没有键按下到 LTHR
        MOV     A,#08H
        AJMP    LKP
LTHR:   JB      ACC.3,NEXT      ;第 4 列没有键按下到 NEXT
        MOV     A,#0CH
        JMP     LKP
NEXT:   MOV     A,R2            ;改变扫描行
        RL      A
        MOV     R2,A
        DJNZ    R1,LK1          ;扫描结束判断
        MOV     KEY,#0FFH
        RET
LKP:    ADD     A,R1            ;计算键值
        DEC     A
        MOV     DPTR,#STAB
        MOVC    A,@A+DPTR       ;查表修改键值
```

```
                CJNE    A,#0AH,KEYH1        ;是 A 键则加入第二功能
                ADD     A,SND
                JMP     KEYEND
KEYH1:          CJNE    A,#0BH,KEYH2        ;是 B 键则加入第二功能
                ADD     A,SND
                JMP     KEYEND
KEYH2:          CJNE    A,#0CH,KEYH3        ;是 C 键则加入第二功能
                ADD     A,SND
                JMP     KEYEND
KEYH3:          CJNE    A,#0DH,KEYH4        ;是 D 键则加入第二功能
                ADD     A,SND
KEYH4:          CJNE    A,#0EH,KEYEND       ;是 E 键则加入第二功能
                ADD     A,SND
KEYEND:         MOV     KEY,A
LK2:            ACALL   DISPLAY
                ACALL   KSI
                JNZ     LK2                 ;按键没松开等待直到按键松开
                RET
STAB:DB 0CH,09H,05H,01H,0DH,00H,06H,02H,0EH,0AH,07H,03H,0FH,0BH,08H,04H
/*  检查是否有键按下子函数，有键按下 A 不为零，反之 A 为零  */
KSI:            MOV     DPTR,#PORTC
                MOV     A,#00H
                MOVX    @DPTR,A
                NOP
                NOP
                MOVX    A,@DPTR
                CPL     A
                ANL     A,#0FH
                RET
DISPLAY:        MOV     R3,#0FH             ;显示子函数
LLPP:           MOV     R0,#OPERAND1        ;显示数据的首地址
                MOV     R1,#00H
                MOV     A,SUMBIT1
                JNZ     NOZ
                ADD     A,#01H              ;没有数据输入时显示一位 0
NOZ:            MOV     R6,A
                MOV     R5,#80H
XIANSHI:        MOV     A,POINT1
                SETB    CY
                SUBB    A,R1
                JZ      DSP1                ;POINT1-R1=0 说明该位要显示小数点
                MOV     DPTR,#TAB1          ;不带小数点的表首地址赋予 DPTR
                JMP     DSP3
DSP1:           MOV     DPTR,#TAB2          ;带有小数点的表首地址赋予 DPTR
DSP3:           MOV     A,@R0
                MOVC    A,@A+DPTR           ;查表得到显示码
                MOVX    @DPTR,A             ;送出位控
                LCALL   YSH2S
                MOV     DPTR,#PORTB
                MOV     A,#00H
                MOV     DPTR,#PORTA
                MOVX    @DPTR,A             ;送出显示码
                MOV     DPTR,#PORTB
                MOV     A,R5
                MOVX    @DPTR ,A            ;关闭显示位，防止鬼影
                INC     R0                  ;修改显示数据的地址
```

```
            INC     R1
            MOV     A,R5            ;修改位控
            RR      A
            MOV     R5,A
            DJNZ    R6,XIANSHI
            DJNZ    R3,LLPP
            RET
YSH2S:      MOV     R7,#0fAH   ;修改这里的参数可以改显示扫描速度快慢
XHD3:       DJNZ    R7,XHD3
            RET
TAB1: DB 3fh,06h,5bh,4fh,66h,6dh,7dh,07h,7fh,6fh,40H,79H
TAB2: DB 0Bfh,86h,0Dbh,0Cfh,0E6h,0Edh,0Fdh,87h,0Ffh,0Efh,40H,79H
            END
```

2）C 语言程序如下：

```
/*本程序实现浮点数（包括正负）加、减、乘、除和开根号的功能，精确到满屏显示，#键*/
/*位第二功能键，按一下实现第二功能按（LED6 亮），再按 led6 灭实现第一功能，数*/
/*字键只有第一功能，ABCDE 键对应第一功能是加、乘、等于、小数点、清零，第二功*/
/*能为减、除、退格、负号、开根号。                                          */
//环境设置选择 LARGE 模式
#include <AT89X51.H>
#include <stdio.h>
#include<absacc.h>
#define PA XBYTE[0x8fff]              //8255 口定义
#define PB XBYTE[0x9fff]
#define PC XBYTE[0Xafff]
#define CR XBYTE[0xbfff]
/*key 值含义 0～9：输入数字；0x0a：+；0x0b：-；0x0c：×；0x0d：÷；0x0e：清零；      */
/*0x0f：第二功能；0x12：=；0x13：退格；0x14：负号；0x15：小数点；0x16：开根号；*/
char key;                              //键码，key=key+snd
struct operand
    {
        char num_bit;                  //操作数总位数
        char num_point;                //操作数小数点后数据位数
        char value[16];                //16 位未压缩 BCD 码表示的操作数
    }operand1,operand2;                //操作数 1 保持输入数据和运算结果
char operand_tmp[16];                  //未压缩 BCD 码表示的 16 位中间结果
char snd;                              //第二功能键标志，按下 F 后 snd=0x07
char operation;                        //操作符，0x0a：+；0x0b：-；0x0c：×；0x0d：÷；
char sign;                             //操作数符号位，7：-；8：+；
char sum_num_bit;                      //总显示位
char max_point;
char bit_xn;
char point_xn;
char LED[11]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x40}; //7 段码显示表
/*********************************************/
/*   延时子程序，延时长短与 t 有关             */
/*********************************************/
delay(int t)
```

```
{int i,j;
 for(i=0;i<t;i++)
    for(j=0;j<t;j++)
     ;
 }
/*****************************************************/
/* 求最大值子函数，返回两数中较大的一个                */
/*****************************************************/
char max(char a,char b)
 {
       if(a>b)
   return a;
   else
   return b;
   }
/****************************************************/
/*   求最小值子函数，返回两数中较小的一个             */
/****************************************************/
 char min(char a,char b)
 {
       if(a>b)
   return b;
   else
   return a;
   }
/*********************************/
/*       显示子函数，显示操作数 1        */
/*********************************/
 void display()
{
int i,j,k;
 char paa;
 char LEDctr[8]={0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01};//显示扫描控制
 if(operand1.num_bit==0)
 k=1;
 else
 k=operand1.num_bit;
 for(i=0;i<3;i++)
    for(j=0;j<k;j++)
    {
         paa=operand1.value[j];
         if(operand1.num_point==j+1)
         PA=LED[paa]+0x80;
         else
         PA=LED[paa];
         PB=LEDctr[j];
         delay(10);
```



```
        PB=0x00;
       }
}
/****************************************************/
/* BCD 转换：将 operand1 转换成 BCD 数，并清零 operand2 */
/****************************************************/
void HEX2BCD()
{
       char i;
for(i=0;i<sum_num_bit;i++)
   {
       if(operand1.value[i]>9)
      {
       operand1.value[i]=operand1.value[i]-10;
       operand1.value[i+1]++;
       }
       operand2.value[i]=0;
   }
     if(operand1.value[sum_num_bit]>0)
        sum_num_bit++;
 }
/*********************************************************/
/*     小数点对齐子函数：加减时使小数点对齐                  */
/*********************************************************/
 void uniform_point()
{
       char i,j;
  char min_point;
       if(operand1.num_point>0)
       operand1.num_point--;
  if(operand2.num_point>0)
     operand2.num_point--;
     max_point=max(operand1.num_point,operand2.num_point);      //小数点最大位数
     min_point=min(operand1.num_point,operand2.num_point);      //小数点最小位数
  sum_num_bit=max(operand1.num_bit-operand1.num_point,operand2.num_bit-
                   operand2.num_point)+max_point;               //总位数最大值
  for(i=0;i<max_point-min_point;i++)
   {
              if(operand1.num_point>operand2.num_point)
               {
                   for(j=operand2.num_bit;j>0;j--)
                 {
                 operand2.value[j+i]=operand2.value[j+i-1];     //operand2 顺次左移 i 位
                 }
             operand2.value[i]=0;                               //末位补 i 个 0
              }
```

```
            else
                {
                    for(j=operand1.num_bit;j>0;j--)
                    {
                    operand1.value[j+i]=operand1.value[j+i-1];          //operand1 顺次左移 i 位
                }
                    operand1.value[i]=0;                                //末位补 i 个 0
                }
        }
}
/***********************************************/
/*   对齐子函数：除法时，数据高位对齐，低位补零*/
/***********************************************/
void uniform_all()
{ char i,j,k;
   char min_bit;
   j=operand1.num_bit;
   k=operand2.num_bit;
   sum_num_bit=max(operand1.num_bit,operand2.num_bit);
   if(operand1.num_bit<operand2.num_bit)                               //operand2 总位数多
    for(i=sum_num_bit-1;j>0;i--)
      {
       operand1.value[i]=operand1.value[j-1];                          //operand1 移位
       j--;
      }
   if(operand1.num_bit>operand2.num_bit)                               //operand1 总位数多
    for(i=sum_num_bit-1;k>0;i--)
      {
       operand2.value[i]=operand2.value[k-1];                          //operand2 移位
         k--;
      }
   min_bit=min(operand1.num_bit,operand2.num_bit);
   min_bit=sum_num_bit-min_bit;
   for(i=0;i<min_bit;i++)                                              //移位后低位补零
    if(operand1.num_bit>operand2.num_bit)
       {
           operand2.value[i]=0;
           operand2.num_point++;
       }
    else
       {
           operand1.value[i]=0;
           operand1.num_point++;
       }
  operand1.num_bit=sum_num_bit;
  operand2.num_bit=sum_num_bit;
  }
```



```
/*********************************************************************************/
/*比较子函数：operand1、operand2 比较大小，operand1 大返回 1，否则返回 0          */
/* 参数 num_bit2、num_bit1 为 operand2、operand1 的位数 （比较时不考虑小数点） */
/*********************************************************************************/
 char compare(char num_bit2,char num_bit1)
{
 if(num_bit2>num_bit1)
 return 0;              //若 operand2 总位数多则 operand2 大
 while(num_bit1>0)
 {
      if(operand1.value[num_bit1-1]>operand2.value[num_bit1-1])
  return 1;            //operand2 最高位大
  if(operand1.value[num_bit1-1]<operand2.value[num_bit1-1])
  return 0;            //operand1 最高位大
  num_bit1--;
  }
  return 0;
 }
/*************************************************************************/
/*   移位子函数：将 operand2（被除数）左移一位，用于除法                    */
/*************************************************************************/
 void rl_1bit()
 {char i=0;
  i=operand2.num_bit;
  while(operand2.num_bit>0)
  {
   operand2.value[operand2.num_bit]=operand2.value[operand2.num_bit-1];
   operand2.num_bit--;
  }
  operand2.value[0]=0;                //前移后末位补 0
  operand2.num_bit=i+1;               //移位后商的位数增 1
  if(operand2.value[i] ==0)           //若 operand2 最高位为 0
   operand2.num_bit--;                //则 operand2 总位数减 1
 }
/***********************************************************************/
/*      结果处理子函数：包括四舍五入、结果位数超出显示和负号处理       */
/***********************************************************************/
 void deal_result(char num_bit,char num_point)
 {char i,j;
  while(operand1.value[num_bit-1] ==0&&num_bit>num_point+1)    //处理最高位没用的零
      num_bit--;
     do{
          if(num_bit>sign)            //判断是否超出显示位数
          {
          i=num_bit-sign;
          num_bit=sign;
```

```
                }
            else
             i=0;
            if(i!=0)                                        //如果超出显示位数则进行四舍五入
             if(operand1.value[i-1]>=5)
                {
                  operand1.value[i]++;
                  num_bit=sign+i;                           //恢复原数位
                    HEX2BCD();                              //未压缩 BCD 进位调整
                    num_bit=num_bit-i;
                  }
            num_point=num_point-i;
            while(operand1.value[i] ==0&&num_point>0)       //处理最低位没用的零
              {
                  num_point--;
                  num_bit--;
                  i++;
                }
            for(j=i;j<num_bit+i;j++)
                   operand1.value[j-i]=operand1.value[j];   //顺次右移
            for(j=0;j<i;j++)
                operand1.value[num_bit+j]=0;
          }                                                 //高位补零
          while(num_bit>sign);
  if(!(num_point>0))
        operand1.num_point=0;
  else
  operand1.num_point=num_point+1;                           //结果小数点个数
  operand1.num_bit=num_bit;                                 //结果总位数
  if(sign==7)                                               //负号处理
        {
        operand1.value[operand1.num_bit]=0x0a;
        operand1.num_bit++;
        }
}
/**********************************************************/
/*   判断数子函数：operand 的所有位为零返回 1，否则返回 0        */
/**********************************************************/
char judge_zero(char*operand,num_bit)
{
        unsigned char i;
        i=num_bit-1;
  for(i=num_bit;i>0;i--)
     if(operand[i-1]!=0)
     return 0;
     return 1;
```



```
}
/***********************************************/
/*  减 1 子函数：operand2 减 operand1 一次，用于除法*/
/***********************************************/
 void sub_chufa()
 {
        char i;
   for(i=0;i<operand1.num_bit;i++)
     {
          operand2.value[i]=operand2.value[i]+22-operand1.value[i];
          if(operand2.value[i]>9)                      //差>9 说明够减
          operand2.value[i]=operand2.value[i]-10;      //将所加的 10 减掉
       else
         operand2.value[i+1]--;                        //若不够减向前借位
     }
    if(operand2.value[operand2.num_bit-1] ==0)
      operand2.num_bit--;                              //将结果最高位无效 0 位数去掉
 }
 /****************************************************/
/*   加法子函数：operand1 加 operand2，结果存入 operand1    */
/****************************************************/
void ADD()
{
        char i;
        uniform_point();
   for(i=0;i<sum_num_bit;i++)
    operand1.value[i]+=operand2.value[i];
   HEX2BCD();
   deal_result(sum_num_bit,max_point);
}
/****************************************************/
/*  减法子函数：operand2 减去 operand1，结果存入 operand1 */
/****************************************************/
void SUBB()
{
        char i;
        char fh;
   uniform_point();
   if(compare(sum_num_bit,sum_num_bit))
    fh=7;                        //operand1 大，结果为负号
   else
    fh=8;                        //operand2 大，结果为正号
 for(i=0;i<sum_num_bit;i++)
   {
        if(fh==8)
         operand1.value[i]=operand2.value[i]+22-operand1.value[i];
```

```
      else                          //+10 先假设有借位
        operand1.value[i]=operand1.value[i]+22-operand2.value[i];
        operand2.value[i]=0;
      if(operand1.value[i]>9)          //结果大于 9 说明不需要借位
        operand1.value[i]=operand1.value[i]-10;
      else if(fh==8)
        operand2.value[i+1]--;
            else
          operand1.value[i+1]--;
    }
  if(fh==sign)
      sign=8;                       //两个负数相减，减数大则结果为正号
                                    //两个正数相减，被减数大则结果为正号
  else
      sign=7;
  deal_result(sum_num_bit,max_point);
}
/****************************************************/
/* 乘法子函数：operand1 乘以 operand2，结果存入 operand1 */
/****************************************************/
void MUL()
{     char i,j;
    if(operand1.num_point>0)
      operand1.num_point--;          //求出 operand1 小数点位数
    if(operand2.num_point>0)
      operand2.num_point--;          //求出 operand2 小数点位数
    /*数值运算*/
    for(i=0;i<operand1.num_bit;i++)
     for(j=0;j<operand2.num_bit;j++)
      {
       operand_tmp[j+i]+=operand1.value[i]*operand2.value[j];  //竖式乘法
            while(operand_tmp[i+j]>9)                          //未压缩 BCD 调整
             {
              operand_tmp[i+j]=operand_tmp[i+j]-10;
              operand_tmp[i+j+1]++;
             }
      }
    operand1.num_bit=i+j-1;                                    //求积总位数
    for(i=0;i<=operand1.num_bit;i++)
     {
       operand1.value[i]=operand_tmp[i];
       operand_tmp[i]=0;
     }
     sum_num_bit=operand1.num_bit+1;
     max_point=operand1.num_point+operand2.num_point;
     deal_result(sum_num_bit,max_point);
```



```
 }
/*******************************************************/
/* 除法子函数：operand2 除以 operand1，结果存入 operand1      */
/*原理：先将两数高位对齐，然后比较大小，operand2 大则        */
/*operand2 减 operand1 一次，商对应位加 1，直到 operand2 小于*/
/*operand1，然后 operand2 左移 1 位并改变商位，开始循环        */
/*******************************************************/
void DIV()
{
 char i,j,k;                    //暂存变量
 char cmp_chufa;                //被除数和除数大小关系标志
 char diff_point;               //被除数和除数小数点之差
 char symbol_point;             //如果 operand1 小数点位数多于 operand2，symbol_point=1;
 if(operand1.num_point>0)
    operand1.num_point--;       //求出 operand1 小数点位数
 if(operand2.num_point>0)
    operand2.num_point--;       //求出 operand2 小数点位数
 while(operand1.value[operand1.num_bit-1]==0)
    operand1.num_bit--;         //去掉 operand1 高位无效 0 的位数
 while(operand2.value[operand2.num_bit-1]==0)
    operand2.num_bit--;         //去掉 operand2 高位无效 0 的位数
    uniform_all();              //operand1 和 operand2 高位对齐
 if(operand1.num_point>=operand2.num_point)
  {
      diff_point =operand1.num_point-operand2.num_point;
  symbol_point=1;
  }
 else
 {
      diff_point=operand2.num_point-operand1.num_point;
  symbol_point=0;
 }
 for(i=0;i<9;i++)               //只能显示 8 位，因此循环 9 次即可
 {
  if(judge_zero(operand2.value,operand2.num_bit))
     break;                     // operand2 为 0 退出
  sum_num_bit=operand1.num_bit;
cmp_chufa=compare(operand2.num_bit,operand1.num_bit);// operand1 和 operand2 比较大小
  while(!cmp_chufa)             // operand2 大则进行减法
  {
   sub_chufa();                 //高 operand1.num_bit 位的 operand2−operand1
   operand_tmp[i]++;            //商相应位加 1
   cmp_chufa=compare(operand2.num_bit,operand1.num_bit);
  }
  rl_1bit();                              //当 operand2 不够减时则移位
 }
```

```
if(i==0)
{
  operand1.num_bit=0;
  operand1.value[0]=0;
  return;
}
if(symbol_point==1)                          //除数小数点位数多
  {
    if(i- diff_point>0)                       //商的数值位数大于小数点差值
      {
            operand1.num_bit=i;               //商的数值位数赋给显存单元
        operand1.num_point=i-1- diff_point;   //商小数点位数
        }
    else
      {
            operand1.num_bit= diff_point+1;   //对齐之后总数值位等于小数点之差加 1
        operand1.num_point=0;
      }
      k=operand1.num_bit;
    for(j=0;j<operand1.num_bit;j++)
      {
          k--;
        operand1.value[k]=operand_tmp[j];
      }
   }
else                                          //除数小数点位数多
  {
        operand1.num_point=i+ diff_point-1;
        operand1.num_bit=operand1.num_point+1;
        k=i;
        for(j=0;j<i;j++)                      //非零数值
            {
             k--;
             operand1.value[k]=operand_tmp[j];
            }
        for(j=i;j<operand1.num_bit;j++)       //非零数值前有 operand1.num_bit−i 个 0
                                              //如 0.0123 非零数值前有 5−3=2 个 0
        operand1.value[j]=0;
   }
  sum_num_bit=operand1.num_bit;
  max_point=operand1.num_point;
  operand2.num_point++;
  deal_result(sum_num_bit,max_point);
 }
/****************************/
/*   检查是否有键按下子函数   */
```



```
/****************************/
 char askey()
{
 PC=0x00;
 delay(3);
 if(PC==0x0f)
 return 0;                        //无键按下返回 0
 else
 return 1;                        //有键按下返回 1
 }
/**********************************/
/*  键盘扫描子函数，将键值存入 key  */
/**********************************/
 char skey()
{
 char i,j,find,ini,inj;
   char atc[4]={0xef,0xdf,0xbf,0x7f};     //键盘扫描控制信号
   char in;
   find=0;
   for(i=0;i<4;i++)
   {
       PC=atc[i];
     delay(3);
     in=PC;
     in=in<<4;
     in=in|0x0f;
         for(j=0;j<4;j++)
             if(atc[j] ==in)
                 {find=1;
                  inj=j;ini=i;
                  }
     }
     if(find==0)
         return 0;
 while(askey())
     display();
     key=ini*4+inj;                 //键码判断
if(key==9)                          //数字处理
     key=0;
else if(key<=9)                     //0～9 为数字
         key++;
if(key>=0x0a&&key<=0x0e)            //a～e 为功能键
   key=key+snd;
return 1;
}
/**************************************/
```

```
/*        存数子函数：将键值存入 operand1    */
/****************************************/
void storage_num()
{char i;
 if(operand1.num_point==0&&operand1.num_bit==0&&key==0)
      return;
 if(operand1.num_bit<8)
      operand1.num_bit++;
 else
      return;
 if(operand1.num_point!=0)
      operand1.num_point++;
 if(operand1.num_bit==1)
 {
       operand1.value[0]=key;return;}
   for(i=operand1.num_bit-1;i>0;i--)
      operand1.value[i]=operand1.value[i-1];
      operand1.value[0]=key;
      return;
 }
/********************************************************/
/*   交换子函数：operand1 的值全部赋给 operand2，operand1 清零  */
/********************************************************/
void exchange()                         //operand1 的值全部赋给 operand2
{
       char i;
 operand2.num_bit=operand1.num_bit;operand2.num_point=operand1.num_point;
 for(i=0;i<operand1.num_bit;i++)
   {
       operand2.value[i]=operand1.value[i];
   operand1.value[i]=0;
   }
   operand1.num_bit=0;operand1.num_point=0;
 }
/**********************/
/*   退格键子函数           */
/**********************/
void back()
{
       char l;
   for(l=1;l<operand1.num_bit;l++)
     operand1.value[l-1]=operand1.value[l];
     if(operand1.num_bit>0)
        operand1.num_bit--;
     operand1.value[operand1.num_bit]=0;
     if(operand1.num_point>0)
```



```
    operand1.num_point--;
}
/************************************************/
/*  除 2 子函数： operand1 除以 2 结果存入 operand1 */
/************************************************/
void DIV2()
{
      char i;
      for(i=operand1.num_bit;i>1;i--)
    {
      operand1.value[i]=operand1.value[i-1]/2;
     if(operand1.value[i-1]%2)
         operand1.value[i-2]+=10;
       }
       operand1.value[1]=operand1.value[0]/2;
       if(operand1.value[0]%2)
         operand1.value[0]=5;
       else
         operand1.value[0]=0;
      sum_num_bit=operand1.num_bit+1;
      if(operand1.num_point==0)
        max_point=operand1.num_point+1;
      else
        max_point=operand1.num_point;
      deal_result(sum_num_bit,max_point);
}
/*****************************************************/
/*  精确子函数：判断开根号是否精确，x[n]=x[n-1]返回 1    */
/*****************************************************/
char precision(char *s2)
{
char i,j;
char point_1;    //x[n]小数位数
char point_2;    //x[n-1]小数位数

 for(i=operand1.num_bit;i<16;i++)
      operand1.value[i]=0;             //运算结果高位清零
 for(i=bit_xn;i<8;i++)
      s2[i]=0;                         //被开根号数高位清零
 if(operand1.num_point>0)
       point_1=operand1.num_point-1;   //求运算结果小数位
 else
       point_1=operand1.num_point;     //求运算结果小数位为 0
 if(point_xn>0)
       point_2=point_xn-1;             //被开根号数小数位
 else
```

```
        point_2=point_xn;                        //被开根号数小数位为 0
        //整数位比较
    sum_num_bit=max(operand1.num_bit-point_1,bit_xn-point_2);
    j=0;
  for(i=sum_num_bit;i>0;i--)
    {
    if(operand1.value[point_1+i-1]!=s2[point_2+i-1])
       return 0;
        j++;
        if(j>6)                                  //6 位相等达到精确条件
       return 1;
    }
     //小数部分比较
    point_1=operand1.num_point;
    point_2=point_xn;
while(point_1>1&&point_2>1)
{if(operand1.value[point_1-2]!=s2[point_2-2])   //小数点后从高到低顺次比较
              return 0;
     point_1--;point_2--;j++;
     if(j>6)
     return 1; }
     if(!(operand1.num_point==point_xn))         //若小数点位数不同则结果不满足要求
          return 0;
          else
              return 1;
}
/*************************************************************/
/*   开根号子函数：牛顿迭代法（x[n]=（y[n-1]/x[n-1]+x[n-1]）/2          */
/*其中 y 是要开根号的数，x1 取一个整数即可                              */
/*************************************************************/
void kaigenghao()               //开根号子函数，牛顿迭代法
{char yn[8],xn[8];
char num_bit_yn;                //y[n-1]总位数
char num_point_yn;              //y[n-1]小数位数
//char num_bit_xn;              //x[n-1]总位数
//char num_point_xn;            //x[n-1]小数位数
  char j,i;
  if(judge_zero(operand1.value,operand1.num_bit))
  {
        operand1.num_bit=0;
    operand1.num_point=0;
    return;
  }
  sign=8;
  if(operand1.num_point==1)//如果小数点个数为 1，则直接赋 0
     operand1.num_point=0;
```

```
for(i=0;i<8;i++)                         //中间变量清 0
{
     yn[i]=0;
     xn[i]=0;
     }
for(i=0;i<operand1.num_bit;i++)
    {
     yn[i]=operand1.value[i];              //被开根号数赋给中间变量
       operand1.value[i]=0;
     }
     num_bit_yn=operand1.num_bit;
     num_point_yn=operand1.num_point;
     bit_xn=1;                             //x[0]总位数
     point_xn=0;                           //x[0]小数点个数
     xn[0]=2;                              //x[0]=2
do {
         for(i=0;i<bit_xn;i++)             //除数 x[n-1]
        operand1.value[i]=xn[i];
     operand1.num_bit=bit_xn;operand1.num_point=point_xn;
     for(i=operand1.num_bit;i<16;i++)  //高位补零
        operand1.value[i]=0;
     for(i=0;i<num_bit_yn;i++)             //被除数 y[n-1]
        operand2.value[i]=yn[i];
     operand2.num_bit=num_bit_yn;
     operand2.num_point=num_point_yn;
    for(i=operand2.num_bit;i<16;i++)   //高位补零
     operand2.value[i]=0;
    for(i=0;i<16;i++)                      //暂存位清零
     operand_tmp[i]=0;
    DIV();                                 //y[n-1]/x[n-1]
    for(i=operand1.num_bit;i<16;i++)
     operand1.value[i]=0;
    for(i=0;i<bit_xn;i++)
     operand2.value[i]=xn[i];
     operand2.num_bit=bit_xn;
     operand2.num_point=point_xn;
       for(i=operand2.num_bit;i<16;i++)
     operand2.value[i]=0;
       ADD();                              //y[n-1]/x[n-1]+x[n-1]
       DIV2();                             //(y[n-1]/x[n-1]+x[n-1])/2
       j=precision(xn);
    for(i=0;i<operand1.num_bit;i++)
      xn[i]=operand1.value[i];
    bit_xn=operand1.num_bit;
    point_xn=operand1.num_point;
    for(i=operand1.num_bit;i<16;i++)
```



```
        operand1.value[i]=0;
      }   while(!j);                        //do-while 结构
    for(i=0;i<bit_xn;i++)
      operand1.value[i]=xn[i];
      operand1.num_bit=bit_xn;
      operand1.num_point=point_xn;
    for(i=0;i<16;i++)                       //清零中间变量为下一次做准备
      operand_tmp[i]=0;
}
/**************************************************/
/*  出错子函数：除数为零时显示 E，清零键复位        */
/**************************************************/
void wrong()                                //错误处理
 {
       char i;
       snd=0;
       P1_7=1;
do{
          PA=0x79;
      PB=0x80;
      while(!askey())
        ;
      skey();
      }while(key!=0x0e);
      for(i=0;i<16;i++)
      {
       operand2.value[i]=0;
            operand1.value[i]=0;
          }
       operand1.num_bit=0;
       operand1.num_point=0;
       operand2.num_bit=0;
       operand2.num_point=0;
      return;
 }
/**************************************************************/
/*  键盘处理子函数：读入键值，并决定下一步的操作                 */
/**************************************************************/
char keywork()
{
       while(1)
   {   while(!skey())
          display();
   if(key>=0&&key<=9)                       //输入数据则记录
         {storage_num();
         continue;
          }
```



```
switch(key)
{case 0x0e: return 1;
 case 0x0f: {
  if(snd==0x07)
        {
         snd=0;
         P1_7=1;}
          else
             {snd=0x07;
              P1_7=0;
             }
             break;
     }
 case 0x11: return 0;
 case 0x12: back();break;
 case 0x13: {if(operand1.value[operand1.num_bit-1]!=0x0a)
                              { operand1.value[operand1.num_bit]=0x0a;operand1.num_bit++;}
                             else{operand1.num_bit--;operand1.value[operand1.num_bit]=0;}
                           }break;
 case 0x14: {if(operand1.num_bit==0)
               operand1.num_bit++;
               if(operand1.num_bit==1&&operand1.value[0]==0x0a)
               {operand1.num_bit++;operand1.value[0]=0;operand1.value[1]=0x0a;}
               if(operand1.num_point==0)
               operand1.num_point=1;}break;
 case 0x15:
{if(!((operand1.num_bit==1&&operand1.value[0]==0)||operand1.num_bit==0))
          {if(operand1.value[operand1.num_bit-1]==0x0a)
             wrong();
           else
             kaigenghao();
          }
       }          break;
 default:if(operation==0)                       //输入操作符
             {operation=key;
                switch(operation)
                {
                   case 0x0a:{ P1_4=0;P1_5=1;P1_6=1;break;}
                   case 0x0b:{ P1_4=1;P1_5=0;P1_6=1;break;}
                   case 0x0c:{ P1_4=1;P1_5=1;P1_6=0;break;}
                   case 0x0d:{ P1_4=0;P1_5=0;P1_6=1;break;}
             }
                if(operand1.num_bit==0)     //operand1 为 0 的情况
                    operand1.num_bit++;
                  exchange();               //operand1 赋给 operand2
              }
```

```
                    else if(operation!=0)
                                 return 0;                    //此处可用于扩展连续运算
                 }

         }
}
/***************************************/
/*   计算子函数：+－×÷运算                */
/* sign   operand1 符号   operand2 符号    */
/* 8              +                 +         */
/* 7              －                －        */
/* 6              －                +         */
/* 5              +                 －        */
/***************************************/
void compute()
{ char i=0;
if(operand1.value[operand1.num_bit-1]!=0x0a&&operand2.value[operand2.num_bit-1]!=0x0a)
          sign=8;
 else if(operand1.value[operand1.num_bit-1]!=0x0a&&operand2.value[operand2.num_bit-1] ==0x0a)
            {operand2.num_bit--;
            operand2.value[operand2.num_bit]=0;
            sign=5;}
  else if(operand1.value[operand1.num_bit-1] ==0x0a&&operand2.value[operand2.num_bit-1]!=0x0a)
            {operand1.num_bit--;
              operand1.value[operand1.num_bit]=0;
              sign=6;}
      else if(operand1.value[operand1.num_bit-1] ==0x0a&&operand2.value[operand2.num_bit - 1] ==
0x0a)
            { sign=7;
             operand2.num_bit--;
             operand2.value[operand2.num_bit]=0;
             operand1.num_bit--;
             operand1.value[operand1.num_bit]=0;}
 if(operation==0x0a)
   {if(sign>=7)
       ADD();
    else   {sign+=2;SUBB();}
    }
 if(operation==0x0b)
   {if(sign>=7)
    SUBB();
    else {sign+=2;ADD();}
    }
 if(operation==0x0c)
    {if(sign>=7)
     sign=8;
```



```
        else
        sign=7;
      MUL();}
    if(operation==0x0d)
      {if(sign>=7)
        sign=8;
        else
        sign=7;
        if(judge_zero(operand1.value,operand1.num_bit))
        wrong();
        else
        DIV();}
     operation=0;                      //操作符清零
     P1_4=1;
     P1_5=1;
     P1_6=1;
     if(operand1.num_bit==1&&operand1.value[0]==0)
      operand1.num_bit--;
     if(operand1.value[1]==0x0a&&operand1.value[0]==0)
      operand1.num_bit--;

     operand2.num_bit=0;               //operand2 总位数存储单元清零
     operand2.num_point=0;             //operand2 小数点位数存储单元清零
     for(i=0;i<16;i++)
        {
          operand2.value[i]=0;         //operand2 数值存储单元清零
          operand_tmp[i]=0;            //操作数暂存单元清零
        if(i>=operand1.num_bit)
          operand1.value[i]=0;
        }
    }
/******************/
/*    主函数        */
/******************/
  void main()
  {char key,i;
     CR=0x81;                          //8255 控制字
     while(1)
       {
       operand1.num_bit=0;
       operand2.num_bit=0;
       operand1.num_point=0;
       operand2.num_point=0;
       snd=0;
       P1_7=1;
       operation=0;
       P1_4=1;
```

```
        P1_5=1;
        P1_6=1;
        for(i=0;i<16;i++)
          {
            operand2.value[i]=0;
            operand_tmp[i]=0;
            operand1.value[i]=0;
          }
        PA=0x3f;
        PB=0x80;
        while(!askey());                //若无键输入等待查询
        while(1)
          {
          if(keywork()==0x00)
            compute();
          else
            break;
          while(!skey())
            display();
            if(key==0x0e)
              break;
          }
        }
    }
```

22.4 案例笔记：电子产品设计步骤

电子产品设计步骤见表 22-9。

表 22-9 电子产品设计步骤

序 号	步 骤	对应本书相关章节
1	确定性能指标，选择方案，结合仿真及实验进行验证	案例 1
2	设计单元电路，计算参数，选择元器件	案例 1
3	画出原理图	案例 2
4	画 PCB 图，制电路板	案例 2
5	元器件焊接	案例 3
6	软件编程	案例 4～7
7	调试	案例 3、案例 8～22
8	装配	
9	文档整理	
10	应用检验	

（1）确定性能指标，选择方案，结合仿真及实验进行验证

1）方案论证和整体设计。方案论证阶段的任务是通过对新产品的设计调研，在产品设

计前突破复杂的关键技术，为确定设计任务书选择最佳设计方案。对于同一个项目，实现的方案可能有多个，实现的途径和技术路线也可能是多方面的。可以将不同的方案和途径加以对比，从中选择一种最优方案来实现。根据电子技术发展的新趋势，寻求将新的技术成果应用于产品设计的途径，有计划地掌握新线路、新结构、新工艺、新理论，以及采用新材料、新器件等，为不断在产品设计中采用新技术，创造出更高水平的新产品奠定基础。



整体设计与方案论证紧密相连。整体设计就是根据设计任务、指标要求和给定条件，设计出符合现场条件的软、硬件方案，并进行方案优化。应划分软、硬件任务，画出系统结构框图。要合理分配系统内部的软、硬资源。选择的原则一般是“相对容易，相对巧妙，性价比高”。主要包括以下四个方面：

① 从系统功能需求出发设计功能模块，包括显示器、键盘、数据采集、检测、通信、控制、驱动、供电方式等。

② 根据设计功能模块分配元器件资源，包括定时/计数器、中断系统、串行口、A-D、D-A、信号调理、时钟发生器等。

③ 从研发条件与市场情况出发选择元器件，包括仿真器、编程器、元器件、编程语言、程序设计等。

④ 从系统可靠性需求确定系统设计工艺，包括去耦、光隔、屏蔽、印制电路板、低功耗、散热、传输距离/速度、节电方式、掉电保护、软件措施等。

硬件研制基本要求如下所述：

① 采用功能强的芯片，以简化电路。

② 硬件资源需留有足够的裕量利于修改和扩展。

③ 自诊断功能，需附加设计有关的监测报警电路。

④ 硬件抗干扰措施。

⑤ 注意印制电路板与机箱、面板的配合，接插件安排等问题，必须考虑安装、调试和维修的方便性。

2）计算机仿真和实验验证。计算机仿真和实验验证的目的是验证电路原理图的正确性、可行性。实验可以分为两种，一是对照原理图进行实物焊接，二是计算机辅助设计（CAD）。计算机辅助设计（CAD）是现在最流行、最快捷和准确的方法，具有下列三项优点：

① 对电路中只能依据经验来确定的元器件参数，用电路仿真的方法很容易确定，而且电路的参数容易调整。

② 电路仿真不受工作场地、仪器设备、元器件品种和数量的限制。

③ 可以大大提高产品的研发速度，降低研发的成本。

随着计算机的普及和 EDA 技术的发展，电子电路设计中的实验演变为仿真和实验的结合。仿真和实验验证要完成以下四项任务：

① 检查各元器件的性能、参数、质量能否满足设计要求。

② 检查各单元电路的功能和指标能否达到设计要求。

③ 检查各个接口电路能否起到应有的作用。

④ 把各单元电路组合起来，检查总体电路的功能、性能是否最佳。

尽管电路仿真有诸多优点，但它仍不能完全代替实验。对于电路中关键部分或采用新技

术、新电路、新器件的部分，一定要进行实验，最终确定可行方案。实验验证还应贯穿元器件选择、原理图、印制电路板绘制、调试等其他环节。

（2）设计单元电路，计算参数，选择元器件

设计单元电路的一般方法和步骤如下所述：

1）根据设计要求和已选定的总体方案原理框图，确定各单元电路的设计要求，拟定主要单元电路的性能指标，分析电路的构成形式。应注意各单元电路之间的相互配合，注意各部分输入信号、输出信号和控制信号的关系。

2）拟定好各单元电路的要求后，按信号流程顺序分别设计各单元电路。

3）从已掌握的知识和了解的各种电路中选择一个合适的电路。若确实找不到性能指标完全满足要求的电路时，也可选用与设计要求比较接近的电路，然后调整电路参数。

4）在单元电路的设计中，特别要注意保证各功能块协调一致地工作。

此外，在进行单元电路设计时，必须明确各单元电路的具体要求，详细拟定单元电路的性能指标，认真考虑各单元之间的相互联系，注意前、后级单元之间信号的传递方式和匹配，尽量少用或不用电平转换之类的接口电路，并考虑到各单元电路的供电电源尽可能统一，以便使整个电子系统简单可靠。另外，尽量选择现有的、成熟的电路来实现单元电路的功能。有时找不到完全满足要求的现成电路，可在与设计要求比较接近的某电路基础上适当改进，或自己进行创造性设计。为了使电子系统的体积小，可靠性高，电路单元尽可能用集成电路组成。

电路设计时除了对电路的性能指标有要求外，通常没有其他任何已知参数，几乎全部由设计者自己选择和计算，这样理论上满足要求的参数值就不会是唯一的，因此需要设计者根据价格、货源等具体情况灵活选择。设计电路中的参数计算，应根据计算值，对参数进行合理选择。一般来说，计算参数应注意以下几点：

1）各元器件的工作电压、电流、频率和功耗等应在允许的范围内，并留有适当的裕量。

2）对于环境温度、交流电网电压等工作条件，计算参数时应按最不利的情况考虑。

3）涉及元器件的极限参数必须留有足够的裕量，一般按约 1.5 倍考虑。

4）对于电阻、电容参数的取值，应选计算值附近的标称值。电阻值一般在 1MW 内选择；非电解电容器一般在 100pF～0.47mF 内选择；电解电容一般在 1～2000mF 范围内选用。

5）在保证电路达到功能指标要求的前提下，尽量减少元器件的品种、价格、体积等。

（3）画出原理图

详见案例 2。

（4）画 PCB 图，制电路板

详见案例 2。

（5）元器件焊接

详见案例 3。

（6）软件编程

程序设计主要包括如下部分：

1）建立数学模型：描述各输入变量和各输出变量之间的数学关系。

2）绘制程序流程图：以简明直观的方式对任务进行描述。

3）程序的编制：注意数据结构、控制算法、存储空间分配，系统硬件资源的合理分配与使用，子程序的I/O口参数的设置与传递。

各程序模块编辑之后，需进行汇编或编译、调试。当满足设计要求后，将各程序模块按照软件结构设计的要求连接起来，即为软件装配。在软件装配时，应注意软件接口。



（7）测试及调试

按照设计任务书规定的设计要求拟定一个测试方案，对各项功能和指标进行逐项测试。如果某项指标不符合要求，需要查明原因，作相应调整直至完全达到设计要求为止。

（8）装配

整机装配工艺就是以设计文件为依据，按照工艺文件的工艺规程和具体要求，把各种电子元器件、机电元器件及结构件装连在PCB、机壳、面板等指定位置上，构成具有一定功能的完整的电子产品的过程。

（9）文档整理

电子设计的总结报告是把设计、组装、调试的内容进行全面总结，而且把实践内容上升到理论高度。总结报告应包括以下几点：

1）课题名称。

2）内容摘要。

3）设计内容及要求。

4）电路设计、参数计算和元器件选择说明。

5）画出完整的电路图，并说明电路的工作原理。

6）组装调试的内容。包括：使用的主要仪器和仪表；调试电路的方法和技巧；测试的数据和波形并与计算结果比较分析；调试中出现的故障、原因及排除方法。

7）总结设计电路和方案的优缺点，指出设计的核心及实用价值，提出改进意见和展望。

8）列出系统需要的元器件。

9）收获、体会。

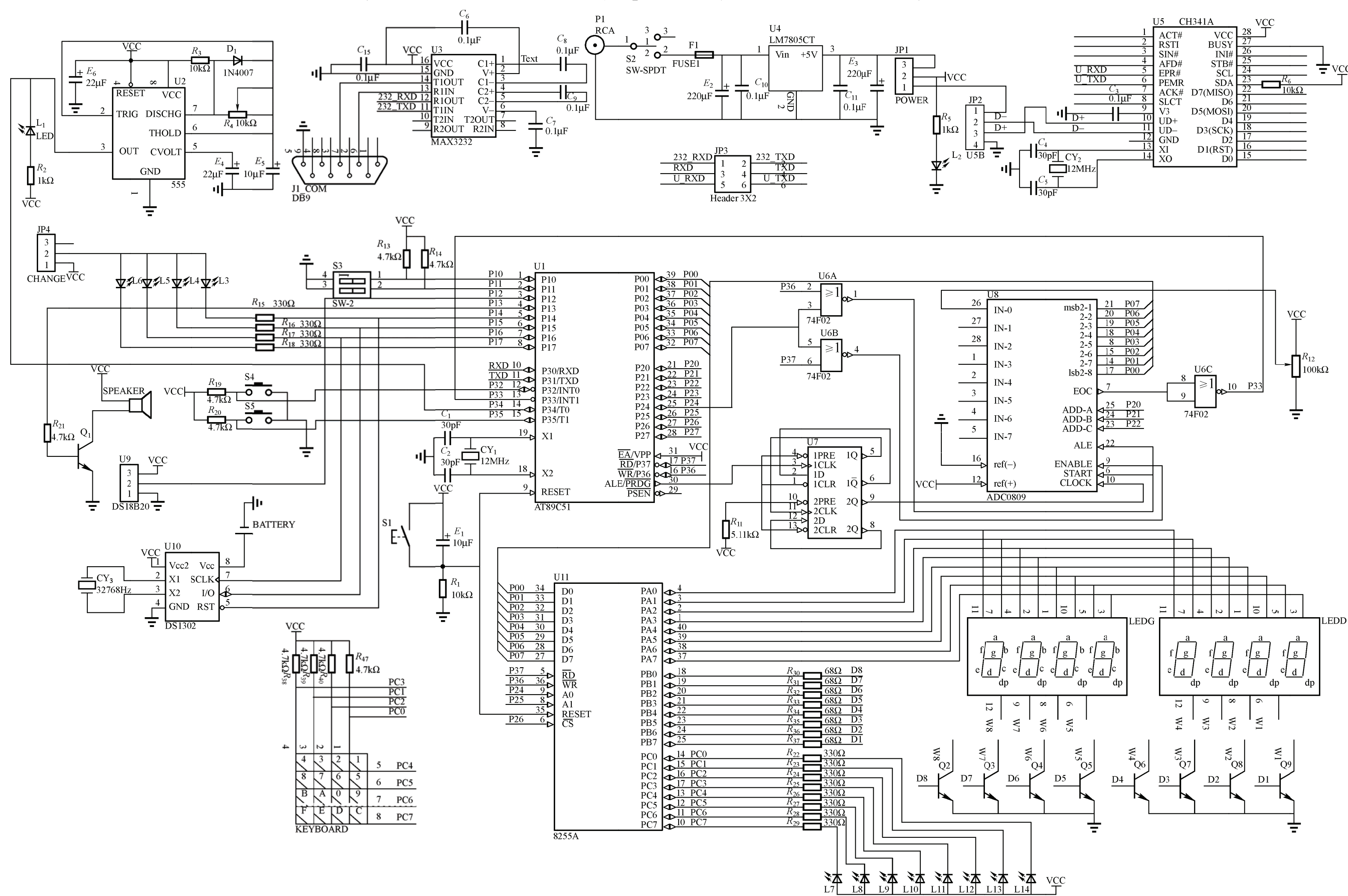
10）参考文献。

总结报告必须做到设计观点、理论分析、方案结论、计算等正确，尽量做到纲目分明、逻辑清楚、内容充实、轻重得当、文字通顺、图样清晰规范。

（10）应用检验

产品的好坏由用户意见决定！根据用户意见进行升级换代才能永葆产品生机！

附录　51 单片机实验板原理图

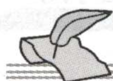


参 考 文 献

[1] 何勤. C 语言程序设计：问题与求解方法[M]. 北京：机械工业出版社，2013.

[2] 陈志旺，李亮. 51 单片机快速上手[M]. 北京：机械工业出版社，2008.

[3] 陈志旺，陈志茹，阎巍山，等. 51 系列单片机系统设计与实践[M]. 北京：电子工业出版社，2009.



本科电气精品教材推荐

普通高等教育电气信息类规划教材

单片机原理与应用

书号：23603　　定价：27.00 元

作者：杭和平　　配套资源：电子教案

推荐简言：

C 语言已经成为单片机开发的主流语言。单片机应用的关键是对单片机功能应用的掌握。实践证明，重点学习高级语言，可以避免在使用汇编语言时，把大量精力花费在局限于具体问题的编程上。本书从实际应用出发，力图从以前单片机教材纠缠具体单片机原理的解析上解脱出来，以 AT89C51 为讲解蓝本，以 C 语言为编程语言，着重讲解单片机各种功能的应用，以及如何用 C 程序去实现要求的功能。

机电传动控制

书号：25733　　定价：33.00 元

作者：张志义　　配套资源：电子教案

推荐简言：

本书以常规机床及数控机床的控制系统为主线，力求突出机电结合、电为机用的特点，从实际应用出发，详细介绍各种电器元件及控制电路。本书共分 8 章，内容包括继电-接触器控制电路的元件、典型环节、常用机床电气控制线路分析；可编程序控制器的基础知识、基本指令系统及设计；伺服电动机原理及驱动、变频器原理及应用；数控机床电气控制系统及数控车床、铣床等电气控制系统介绍。

微机原理与接口技术

书号：33594　　定价：36.00 元

作者：周鹏　　配套资源：电子教案

推荐简言：本书更加适应新形势下高校对微机接口知识的需求，以适应众多工科不同层次、不同类型、不同专业的需要。教师在授课过程根据不同专业和教学大纲要求，因地制宜，灵活使用教材内容。特色是：突出重点，循序渐进，力求通俗易懂；例题丰富，形式多样；注重实用，强调实践性，书中各章都有例题和习题，通过练习可帮助学生将理论知识应用于实践。

TMS320C54X DSP 应用技术教程

书号：35536　　定价：39.80 元

作者：叶青　　配套资源：电子教案

推荐简言：本书以美国 TI 公司的 TMS320C54x 系列 DSP(数字信号处理器)为描述对象，从初学者的角度入手，对 DSP 系统所涉及的硬件和软件技术进行了系统的介绍。本书内容新颖全面、通俗易懂、实用性强，可作为高等院校电子信息、通信、自动化、电气及相关专业、高年级本科生和研究生的教材和参考用书。

PLC 基础及应用教程（三菱 FX2N 系列）

书号：32329　　定价：27.00 元

作者：秦春斌　　配套资源：电子教案

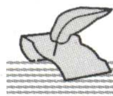
推荐简言：本书在简单介绍电气控制基本电路的基础上，阐述了三菱 FX2N PLC 的结构、工作原理、内部资源及硬件组态配置。然后重点介绍了三菱 FX2N PLC 指令系统、编程规则、编程软件、特殊功能模块和通信模块及其应用。最后，通过 PLC 控制系统设计方法和 PLC 在工程中的应用实例，对常用 PLC 控制系统的设计思想、设计步骤、设计方法及调试维护，进行了详细讲述。

传感器技术实用教程

书号：35962　　定价：38.00 元

作者：吕勇军　　配套资源：电子教案

推荐简言：本书对于每种传感器，在阐述基本工作原理的基础上，均给出了典型测量电路和应用实例。本书特色是：以被测对象为线索介绍相关传感器，便于读者掌握、比较与选择传感器；简化工作原理以及工艺结构的描述，强化传感器的外部特性、主要参数、接口方式以及应用电路等方面内容，可帮助读者在了解传感器工作原理的基础上，掌握选择合适传感器和正确使用传感器的方法。



本科电气精品教材推荐

21 世纪高等院校电气信息类系列教材

FX 系列 PLC 编程及应用

书号：16219　**定价：**29.00 元

作者：廖常初　**配套资源：**光盘

推荐简言：

经典畅销书，累计销量 8 万余册。本书以三菱的 FX 系列 PLC 为例，介绍了 PLC 的工作原理、硬件结构、编程元件与指令系统，还介绍了梯形图的经验设计法、继电器电路转换法和顺序控制设计法，这些编程方法易学易用，可以节约大量的设计时间。

S7-200PLC 编程及应用

书号：21650　**定价：**32.00 元

作者：廖常初　**配套资源：**电子教案、光盘

获奖情况：普通高等教育“十一五”国家级规划教材

推荐简言：

本书是全国优秀畅销书《PLC 编程及应用》 教材版，以西门子公司的 S7-200 为例，介绍了 PLC 的工作原理、硬件结构、指令系统、最新版编程软件和仿真软件的使用方法。各章配有习题，附有实验指导书和部分习题的答案。本书配套的光盘有 S7-200 编程软件和 OPG 服务器软件 PCAccess、与 S7-200 有关的中英文手册和应用例程等。

微型计算机原理与接口技术 第 2 版

书号：26218　**定价：**37.00 元

作者：张荣标　**配套资源：**电子教案

推荐简言：经典畅销书。本书以 Intel 系列微处理器为背景，介绍了微型计算机原理与接口技术。全书以弄懂原理、掌握应用为编写宗旨。本书分三个部分：微型计算机原理部分，汇编语言程序设计部分，接口与应用部分。

单片机原理与应用 第 2 版

书号：26506　**定价：**36.00 元

作者：赵德安　**配套资源：**电子教案

获奖情况：普通高等教育“十一五”国家级规划教材

推荐简言：本书全面系统地讲述了 MCS-51 系列单片机的基本结构和工作原理、基本系统、指令系统、汇编语言程序设计、并行和串行扩展方法、人机接口、以及单片机的开发应用等方面的内容，每章都附有习题，以供课后练习。

微型计算机控制技术

书号：28859　**定价：**27.00 元

作者：黄勤　**配套资源：**电子教案

推荐简言：本书共分 7 章，以 80X86 及 51 系列单片机为控制工具，其主要内容包括：微型计算机控制系统的一般概念；系统设计的基本内容和方法；工业控制微型计算机的过程输入输出技术、数据通信技术、控制网络技术、现场总线技术、分散型控制系统（DCS）的构成、工控组态软件的设计思想及相关包的使用方法。

集散控制与现场总线（第 2 版）

书号：34393　**定价：**18.00 元

作者：刘国海　**配套资源：**电子教案

推荐简言：本书将控制领域的两大技术热点——集散控制和现场总线有机结合起来，从集散控制系统的基本思想、硬件软件体系等方面进行了介绍。介绍了集散控制系统的通信系统、控制算法设计评价等相关技术，全面分析了 ControlNet、DeviceNet、Profibus、FF、CAN 等现场总线技术特点、通信接口的设计方法，并给出了工程应用举例。



本科电气精品教材推荐

电气信息工程丛书

西门子工业通信网络组态编程与故障诊断

书号：28256 **定价：**69.00 元

作者：廖常初 **配套资源：**DVD 光盘

推荐简言：

本书建立在大量实验的基础上，详细介绍了实现通信最关键的组态和编程方法，随书光盘有上百个通信例程，绝大多数例程经过硬件实验的验证。读者根据正文介绍的通信系统的组态步骤和方法，参考光盘中的例程作组态和编程练习，可以较快地掌握网络通信的实现方法。

西门子 PLC 高级应用实例精解

书号：29304 **定价：**42.00 元

作者：向晓汉 **配套资源：**DVD 光盘

推荐简言：

本书通过实例全面讲解西门子 S7-200/S7-1200/S7-300 PLC 的高级应用。内容包括梯形图的编程方法、PLC 在过程控制中应用、PLC 在运动控制中的应用、PLC 的通信及其通信模块的应用等。书中实例都用工程实际的开发过程详细介绍，便于读者模仿学习。每个实例都有详细的软件、硬件配置清单，并配有接线图和程序。本书所附配套资源中有重点实例源程序和操作过程视频文件。

西门子 WinCC V7 基础与应用

书号：32902 **定价：**44.00 元

作者：甄立东 **配套资源：**DVD 光盘

推荐简言：本书系统地介绍了 WinCC V7.0 的功能及其组态方法。首先介绍了初级用户必须掌握的主要功能，其次介绍了高级用户需要了解的 Microsoft SQL Server 2005、冗余系统组态、全集成自动化、开发性和工厂智能选件。通过实例，详尽地展示了各种应用的设计和实现步骤以及应用。本书还对 WinCC V7.0 新增功能进行了详细讲解。

现场总线与工业以太网及其应用技术

书号：35607 **定价：**58.00 元

作者：李正军 **配套资源：**电子教案

推荐简言：本书从工程实际应用出发，全面系统地介绍了现场总线与工业以太网技术及其应用系统设计，力求所讲内容具有较强的可移植性、先进性、系统性、应用性、资料开放性，起到举一反三的作用。主要内容包括现场总线与工业以太网概论、控制网络技术、通用串行通信接口技术、PROFIBUS 现场总线、PROFIBUS-DP 通信控制器与网络接口卡等。

PLC 编程及应用 第 3 版

书号：10877 **定价：**37.00 元

作者：廖常初 **配套资源：**DVD 光盘

获奖情况：全国优秀畅销书

推荐简言：西门子公司重点推荐图书，累计销量已达 **12** 万册。本书以西门子公司的 **S7-200 PLC** 为例，介绍了 **PLC** 的工作原理、硬件结构、指令系统、最新版编程软件和仿真软件的使用方法；介绍了数字量控制梯形图的一整套先进完整的设计方法；介绍了 **S7-200** 的通信网络、通信功能和通信程序的设计方法等。配套光盘有 **S7-200** 编程软件和 **OPC** 服务器软件 **PC Access**、与 **S7-200** 有关的中英文用户手册和资料、应用例程等

S7-300/400 PLC 应用技术 第 3 版

书号：36379 **定价：**69.00 元

作者：廖常初 **配套资源：**DVD 光盘

推荐简言：西门子公司重点推荐图书，销量已达 8 万册。本书介绍了 S7-300/400 的硬件结构、性能指标和硬件组态的方法；指令系统、程序结构、编程软件 STEP7 的使用方法；梯形图的经验设计法、继电器电路转换法和顺序控制设计法，以及使用顺序功能图语言 S7 Graph 的设计方法。另外还介绍了 S7-300/400 的网络结构，AS-i 和工业以太网、PRODAVE 通信软件的组态、参数设置的编程的方法。配套的光盘附有大量的中英文用户手册、软件和例程，附有 STEP7 编程软件。

39

在线互动交流平台

官方微博：http://weibo.com/cmpjsj
豆 瓣 网：http://site.douban.com/139085/
读者信箱：cmp_itbook@163.com



51单片机案例笔记

作者简介



陈志旺 燕山大学副教授，长期从事嵌入式系统基础教学、研究、写作工作，主持河北省自然科学基金1项，专利2项，发表文章30余篇。指导学生曾获“挑战杯”国家级三等奖1项(2011)，“挑战杯”河北省一等奖1项(2011)；主编《51单片机快速上手》、《51系列单片机系统设计与实践》、《STM32嵌入式微控制器快速上手(第1、2版)》等教学辅导书，其中《STM32嵌入式微控制器快速上手(第2版)》荣获2014年中国电子教育学会“全国电子信息类优秀教材”三等奖。

地址：北京市百万庄大街22号
邮政编码：100037
电话服务
服务咨询热线：010-88379833
读者购书热线：010-88379469
网络服务
机工官网：www.cmpbook.com
机工官博：weibo.com/cmp1952
金书网：www.golden-book.com
教育服务网：www.cmpedu.com
封面无防伪标均为盗版



机械工业出版社
微信服务号



计算机分社微信服务号

上架指导 工业技术/单片机

ISBN 978-7-111-49736-3

策划编辑◎时 静 / 封面设计◎





定价：49.00元